DIPLOMAMUNKA

Vas Gergely

Debrecen

2009

Debreceni Egyetem

Informatika Kar

Elosztott rendszerek deployolási problémái

Belső témavezető:

Dr. Végh János

Külső konzulens:

Hajdu Tamás

ROC Development

Hungary Kft

Készítette:

Vas Gergely

Programtervező Matematikus

Debrecen

2009

TARTALOMJEGYZÉK

BEVEZETÉS

A számítástechnika kezdetén még csak egyetlen számítógépen futottak a programok. Az idő előrehaladtával az erőforrások megosztása és a könnyebb információcsere érdekében kialakultak a helyi és nagy távolságú számítógépes hálózatok. Az Internet elterjedésével egyre növekvő terhelést már nem tudták a korábbi szerverek kiszolgálni, csak ha egyre drágább eszközöket szereltek beléjük, vagy egyre több szervert állítottak munkába. Ezen eszközök teljesítménye azonban nem volt annyival nagyobb az olcsóbb eszközökhöz képest, mint amennyivel több pénzbe kerültek.
Ez vezetett oda, hogy az elosztott rendszerek létrejöttek. A szerverek funkcionalitását hálózati alapokra helyezték, amely hálózatban a számítógépek együtt dolgoznak a hozzájuk érkező igények kiszolgálása érdekében. Az igények kielégítése úgy történik, hogy a felhasználó ezt a hálózatot egyetlen számítógépnek látja. Mivel az igények kiszolgálását több számítógép végzi, így ez nagyobb megbízhatóságot, rendelkezésre állást és elérhetőséget biztosít. Gondoljuk csak el, hogy mekkora károkat okozhatna például egy tőzsdei rendszer esetén - ahol hatalmas pénzmozgások vannak - ha az akár csak néhány percre is leállna. Ha viszont elosztott rendszer segítségével történik a megvalósítás, akkor egy vagy több számítógép kiesése sem okoz gondot, hiszen a többi gép is el tudja látni az igények kiszolgálását.

A C# nyelvvel 2007 telén kezdtem el ismerkedni, autodidakta módon. Ezzel párhuzamosan lehetőségem nyílt az ROC Development Hungary Kft-nél, Hajdu Tamás és Kern Péter vezetésével egy program megírására. A program a következő probléma megoldását célozza meg: A munkaidő csekély részében nyílik lehetőség új alkalmazások számítógépre történő telepítésére. Gyakran előfordul azonban, hogy csak a kiválasztott alkalmazás telepítése után szembesülünk azzal, hogy a program megfelelő működéséhez további termékek feltelepítése szükséges. Ezen probléma megoldását, és ezáltal a rendelkezésre álló idő hatékonyabb kihasználását, céloztam meg a ProDep nevű alkalmazás segítésével.

Elosztott rendszer

„Akkor tudhatod, hogy ilyennel dolgozol, ha egy általad soha nem hallott számítógép meghibásodása megakadályoz téged bármiféle munka elvégzésében”(Leslie Lamport)(1)

Az elosztott rendszer úgy képzelhető el, mint önálló számítógépeknek egy olyan együttese, amely a felhasználó számára egyetlen számítógépnek látszik. Mindezt úgy oldják meg, hogy elfedik a számítógépek közötti különbségeket, a gépek közötti kommunikációt és a rendszer belső struktúráját (1).
Az elosztott rendszer négy fő célja, hogy a felhasználó számára elérhetővé tegyen távoli erőforrásokat. Ilyen erőforrás lehet például egy nyomtató, de adatbázis is tárolható elosztott módon, illetve léteznek elosztott fájlrendszerek is, a világ legnagyobb elosztott rendszeréről, az Internetről nem is beszélve (1).

Elosztott rendszerek átlátszósága

Azt az elosztott rendszert, ami a felhasználók vagy alkalmazások számára úgy viselkedik, mintha egyetlen számítógép lenne, átlátszónak nevezzük. Most tekintsük át hogy, átlátszóságnak milyen fajtái léteznek (1)!

a) Hozzáférhetőségi átlátszóság
b) Elhelyezkedési átlátszóság
c) Áthelyezhetőségi átlátszóság
d) Mozgathatósági átlátszóság
e) Többszörözhetőségi átlátszóság
f) Egyidejűségi átlátszóság
g) Meghibásodási átlátszóság
h) Állandósági átlátszóság

A hozzáférhetőségi átlátszóság azt határozza meg, hogy az erőforrások milyen módon érhetőek el, illetve az adatábrázolási különbségek hogyan maradnak rejtve (az önálló számítógépek processzorainak különbözőségéből adódik a *„little endian”* ábrázolás és a *„big endian”* ábrázolás közötti különbség).

Az elhelyezkedési átlátszóság foglalkozik azzal a kérdéssel, hogy a felhasználó nem ismerheti az erőforrás fizikai helyét az elosztott rendszeren belül.
Az áthelyezhetőségi átlátszóság akkor teljesül egy elosztott rendszerre, ha az erőforrások úgy helyezhetőek át máshova, hogy ez nem változtat semmit az elérésükön. Speciális fajtája, a mozgathatósági átlátszóság. Ekkor az erőforrás használat közben is áthelyezhető, és az áthelyezés közben a felhasználók vagy alkalmazások semmi változást nem tapasztalnak.
A többszörözhetőségi átlátszóság akkor áll fenn, ha az erőforrás többszörözése rejtett a felhasználók előtt.
Az egyidejűségi átlátszóság biztosítja a felhasználó számára azt az érzetet, hogy a megosztott erőforrást kizárólag ő használja.
A meghibásodási átlátszóság esetén a felhasználók nem veszik észre, hogy a rendszer egy erőforrása nem elérhető, vagy valamilyen korábbi hiba kijavítása után újra elérhetővé válik.
Az állandósági átlátszóság akkor teljesül, ha elrejtjük, hogy az erőforrás a memóriában vagy a háttértáron helyezkedik-e el. (1)

Elosztott rendszerek hibatűrése

Elosztott rendszerek esetén felmerül a részleges meghibásodás lehetősége. Ez azt jelenti, hogy nem az egész elosztott rendszer hibásodik meg, hanem annak csupán egy vagy néhány összetevője. Ennek a meghibásodásnak a következtében előfordulhat további összetevők meghibásodása is, de maradhatnak olyan összetevők, amelyek változatlanul működnek tovább. Amikor meghibásodás következik be, akkor az elosztott rendszernek úgy kell tovább működnie, hogy azt a felhasználók, alkalmazások ne vegyék észre. Ezt nevezzük *meghibásodási átlátszóságnak* (1).

Folyamatok rugalmassága

A hibás folyamat kiküszöbölésére több azonos folyamatot csoportba szoktak szervezni. A csoportokat belső felépítésük alapján különböztethetjük meg. Bizonyos csoportoknál minden folyamat egyenlő, míg más csoportok hierarchikus felépítésűek. Az *egyenlő csoport* szimmetrikus és nincs hibára érzékeny pontja. Ha egy folyamat összeomlik, a csoport ugyan

kisebb lesz, de a működése zavartalanul folyik tovább. Az ilyen csoportok hátránya, hogy a döntéshozatal bonyolultabbá válik, mivel gyakran szavazniuk kell a döntés eredményéről (1).
A *hierarchikus csoport* esetén az egyik folyamatot kinevezzük *koordinátor*nak, a többi folyamat pedig a *dolgozó* lesz. Ezen modellnél, ha egy munkát el kell végezni, akkor a koordinátor dönti el, hogy melyik folyamathoz továbbítja a kérést, melyik folyamat végezze el a munkát. A modell hátránya, hogy a koordinátor összeomlása a teljes csoport működésképtelenségéhez vezet, de amíg az működik, addig a döntéseket az hozza meg egyedül, nincs szükség szavazásra (1).

Elosztott véglegesítés

Az *elosztott véglegesítés* problémája akkor merül fel, ha egy műveletet egy csoport minden tagjának végre kell hajtania, vagy ellenkezőleg, egyetlen tagjának sem kell végrehajtania. A művelet megbízható csoportcímzés esetén az üzenet kézbesítésének feleltethető meg, elosztott tranzakció esetén a művelet a tranzakció adott, a tranzakcióban részt vevő egyetlen helyen való lezárása lehet (1). A művelet lehet akár egy elosztott rendszerbeli alkalmazás telepítése, deploy-olása is.
Az elosztott véglegesítés gyakran koordinátor segítségével történik. Ha a koordinátor mondja meg a műveletben érintett többi folyamatnak (melyeket résztvevőknek is nevezünk), hogy helyileg végre kell-e hajtaniuk a kérdéses műveletet vagy sem, akkor **egyfázisú véglegesítési protokollról** (one-phase commit protocol) beszélünk. A módszer nagy hátránya, hogy nincs visszajelzés a koordinátor felé arról, hogy az egyes résztvevők végre tudták-e hajtani a műveletet vagy sem. Éppen ezért a gyakorlatban ennél egy jobb, megbízhatóbb módszerre van szükség (1).

Kétfázisú véglegesítési protokoll (2PC – two-phase commit protocol)

A gyakorlatban talán a leggyakoribb módszer az elosztott véglegesítésre a *kétfázisú véglegesítési protokoll*. A következőkben egy olyan elosztott tranzakcióval lesz szemléltetve a módszer, amelyben részt vesznek olyan folyamatok, melyek mindegyike más-más gépen fut (1). Ez a példa átültethető egy alkalmazás, elosztott rendszer több gépére való, egyidejű telepítésére, deploy-olására.

A módszer, mint ahogy arra a neve is utal, a véglegesítést két fázisban végzi el:

1. Szavazási fázis
    a. A koordinátor valamennyi résztvevőnek VOTE_REQUEST üzenetet küld.
    b. Amikor egy résztvevő megkapja a VOTE_REQUEST üzenetet, akkor vagy a VOTE_COMMIT üzenet segítségével tudatja a koordinátorral, hogy felkészült a tranzakció helyi véglegesítésére, vagy VOTE_ABORT üzenetet küld vissza, jelezve, hogy nem tudja a véglegesítést elvégezni.
2. Döntési fázis
    a. A koordinátor begyűjti az összes szavazatot a résztvevőktől. Ha mindegyik résztvevő VOTE_COMMIT üzenetet küldött vissza, akkor a koordinátor a véglegesítés mellett dönt, és minden résztvevőnek elküld egy GLOBAL_COMMIT üzenetet. Ha csak egy résztvevő is VOTE_ABORT üzenettel válaszolt, akkor a koordinátor a félbeszakítás mellett dönt és egy GLOBAL_ABORT üzenet lesz elküldve.
    b. Minden résztvevő, amely a véglegesítésre szavazott, várakozni kezd a koordinátor végső döntésére. Ha a résztvevő később GLOBAL_COMMIT üzenetet kapja, akkor helyileg végrehajtja a véglegesítést, a GLOBAL_ABORT üzenetet hatására azonban a tranzakció helyileg félbe lesz szakítva (1).

Ha ezt a 2PC-protokollt olyan környezetben valósítjuk meg, ahol előfordulhat meghibásodás, akkor problémát jelent, hogy mind a koordinátornak, mind a résztvevőknek van olyan állapotuk, amelyben a bejövő üzenet érkezéséig blokkolódnak. Probléma akkor fordul elő, ha egy folyamat összeomlik, ekkor ugyanis az innen érkező üzenetre váró többi folyamat a végtelenségig várakozó állapotban marad. Összesen három olyan állapot van, amelyben vagy a koordinátor, vagy a résztvevő az üzenet megérkezéséig blokkolva marad:

1. Először is a résztvevő saját INIT állapotában a koordinátor VOTE_REQUEST üzenetére várakozhat. Ha ez az üzenet egy adott ideig nem érkezik meg, akkor a résztvevő egyszerűen helyileg félbeszakítja a tranzakciót, majd VOTE_ABORT üzenetet küld a koordinátornak.

2. A koordinátor saját WAIT állapotában blokkolódhat, miközben a résztvevők szavazatára várakozik. Ha egy bizonyos időtartamon belül nem érkezik be az összes szavazat, a koordinátornak a tranzakció félbeszakítására szavaz, majd GLOBAL_ABORT üzenetet küld

a résztvevőknek.

3. Végül a résztvevők READY állapotban blokkolódhatnak, amikor a koordinátor végső döntésére várakoznak. Ha ez az üzenet nem érkezik meg adott időn belül, a résztvevő nem dönthet saját hatáskörben a tranzakció félbeszakításáról, hanem ki kell találnia a koordinátor döntését. Ekkora az egyik megoldás, hogy minden résztvevő várakozik, amíg a koordinátor újra nem indul. Egy másik, ha a P résztvevőnek megengedjük, hogy felvegye a kapcsolatot a Q résztvevővel, és megkísérelje annak állapotából kitalálni, hogy mit is kell tennie:

    - Tegyük fel, hogy Q már elérte a COMMIT állapotot. Ez csak akkor lehetséges, ha a koordinátor még összeomlása előtt elküldte a GLOBAL_COMMIT üzenetet Q-nak, amelyet P már nem látott. Ekkor P is véglegesítheti helyileg a tranzakciót.
    - Ugyanígy ha Q már ABORT állapotban van, akkor P félbeszakíthatja helyileg a tranzakciót.
    - Tegyük fel, hogy Q INIT állapotban van. Ez akkor következhet be, ha a koordinátor elkezdte kiküldeni a VOTE_REQUEST üzeneteket, amelyeket P meg is kapott (és elküldte a VOTE_COMMIT üzenetet), Q azonban még nem kapta meg. Ebben az esetben mind P, mind Q átmehet ABORT állapotba.
    - A legbonyolultabb helyzet akkor következik be, ha Q is a READY állapotban van, a koordinátor válaszát várva. Előfordulhat, hogy valamennyi résztvevő READY állapotban várakozik – ebben az esetben nem lehet meghozni a döntést. A baj az, hogy bár valamennyi résztvevő véglegesíteni akarja a tranzakciót, ehhez a koordinátor „engedélyére” van szükség. Tehát a protokoll a koordinátor újraindulásáig blokkolódik. Ezért nevezzük a 2PC-t még blokkoló véglegesítési protokollnak (blocking commit protocol) is. Mivel a 2PC blokkolódása csak igen ritkán következik be, ezért a gyakorlatban ez a módszer terjedt el leginkább (1).

Ahhoz, hogy a folyamat kijavíthassa magát, saját állapotát állandó tárolóra kell mentenie.

- Így például ha a résztvevő INIT állapotban volt, akkor felépülése után biztonságosan elhatározhatja a tranzakció helyi félbeszakítását, amiről tájékoztatja a koordinátort.
- Ugyanígy ha a döntést már korábban sikerült meghoznia, vagyis COMMIT vagy ABORT állapotban volt összeomláskor, akkor egyszerűen ismét felveszi ezt az állapotot, és döntését újra közli a koordinátorral.

- A gondok akkor jelentkeznek, ha a résztvevő READY állapotban omlott össze. Ebben az esetben felépülésekor nem képes egyedül eldönteni, hogy véglegesíteni vagy félbeszakítani kell-e a tranzakciót, így rákényszerül, hogy felvegye a kapcsolatot egy másik résztvevővel, ahhoz hasonlóan, ahogy az imént a READY állapotba beragadt résztvevő tette (1).

A koordinátornak csak két kritikus állapotot kell nyomon követnie:

- Amikor megkezdi a 2PC-protokollt, fel kell jegyeznie, hogy WAIT állapotba készül belépni, s így felépülése után újra el tudja küldeni a VOTE_REQUEST kérést az összes résztvevőnek.
- Ugyanígy ha a második fázisban meghozta döntését, akkor elegendő ezt a döntést feljegyezni, hogy az a felépüléskor újra elküldhető legyen (1).

Alkalmazás telepítése Oracle WebLogic segítségével

Ha az Oracle WebLogic környezetben szeretnénk egy alkalmazás telepíteni a klaszterre, akkor a következő módszerek közül választhatunk:

- *WebLogic Server Administration Console*: Ez a konzol a BEA adminisztrációs szolgáltatás grafikus felhasználói felülete.
- *weblogic.Deployer*: A webLogic.Deployer egy Java alapú segédprogram, amely egy parancssori interfészt nyújt a WebLogic Server telepítési API-hoz.
- *WebLogic Scripting Tool (WLST)*: Ez a lehetőség is egy parancssori interfész, amely domain konfigurációs feladatok automatizálására használható, beleértve az alkalmazástelepítési beállításokat és a telepítési műveleteket (2).

Függetlenül a használt telepítési eszköztől, amikor elindul a telepítési folyamat, meghatározásra kerülnek a telepíteni kívánt komponensek, valamint a telepítés helye- egy klaszter vagy egy klaszteren, esetleg domain-en belüli szerver. A domain adminisztrációs szervere irányítja a telepítési folyamatot és egész idő alatt kommunikál a klaszter Managed Server-eivel. Minden Managed Server letölti azokat a komponenseket, amiket telepíteni kell és elindítják a helyi telepítési feladataikat. A telepített komponensek telepítési állapotát az aktuális MBeans kezeli. A WebLogic szerverre történő telepítés előtt a telepíteni kívánt alkalmazásokat be kell csomagolni (2).

A WebLogic szerveren az alkalmazásokat két fázisban telepíthetjük. Mielőtt a telepítés

elkezdődne a WebLogic Server meghatározza a klaszter Managed Server-einek az elérhetőségét.

A telepítés első fázisában az alkalmazás komponenseit kiosztják a kijelölt szerverpéldányoknak és a tervezett telepítést validálják annak érdekében, hogy meggyőződjenek arról, hogy a telepítés sikeresen végrehajtható. Ezen fázis alatt a telepítés alatt álló alkalmazásra irányuló felhasználói kérések nincsenek engedélyezve. A kiosztási és validálási folyamat alatt felmerülő hibák a telepítés felfüggesztését eredményezik minden szerverpéldányon – még azokon is, amelyeken a validálás sikeres volt. Az előkészített fájlok nem lesznek eltávolítva, ugyanakkor az előkészítés alatt végrehajtott helyi változások visszavonásra kerülnek.

Miután az alkalmazás komponenseit eljuttatták a célállomásokra és validálták őket, elkezdődik a második fázis. Ekkor az alkalmazást feltelepítik a célként kijelölt szerver példányokra és a telepített alkalmazás elérhetővé válik a kliensek számára. Ha a második szakaszban hiba lép fel, a szerver az alábbi módokon viselkedhet induláskor:

- ha telepítés közben a szerver példányon hiba következik be, akkor az ADMIN módban fog indulni.
- ha egy klaszter tagja nem tudja telepíteni az alkalmazást, akkor az alkalmazás elérhetetlenné válik (2).

A klaszterre való telepítés irányelvei

Ideális esetben az összes Managed Server fut és elérhető a telepítési folyamat során. Nem javasolt úgy telepíteni egy alkalmazást, hogy a klaszter néhány tagja nem elérhető. Mielőtt egy alkalmazást telepítenénk egy klaszterre, gondoskodjunk, hogy a klaszter összes Managed Server-e fusson és elérhető legyen az adminisztrációs szerver által. Ha egy olyan Managed Serverre telepítünk egy alkalmazást, ami a telepítés alatt le van választva a klaszterről – fut, de az adminisztrációs szerver nem éri el – problémát fog okozni, amikor az a szerver újra csatlakozik a klaszterhez és megpróbálnak hozzá csatlakozni. A szinkronizációs folyamat alatt, miközben a többi klaszterben lévő Managed Server újra felépíti a kapcsolatot az előbb még leválasztott szerverpéldánnyal, a telepített alkalmazásra irányuló felhasználói kérések és a másodlagos session-ök kialakítására vonatkozó kérések a szerver példányra sikertelenek lesznek. Ezen állapot előfordulási kockázatát csökkenteni lehet a ClusterConstraintsEnabled

beállítással.

Másik fontos tanács, hogy a klaszter tagság ne változzon a telepítési folyamat során. Azaz miután egy telepítés elindult, ne tegyük a következőket:

- Ne adjunk hozzá vagy távolítsunk el egy vagy több Managed Server-t a célklaszterből.
- Ne állítsunk le egy vagy több célklaszterbeli Managed Server-t (2).

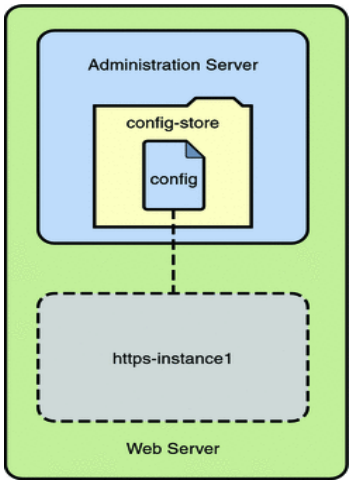
A WebLogic Server korábbi verziói a következőképpen támogatták e helyzet elkerülését:

- Ha egy klaszter egy vagy több Managed Server-e nem volt elérhető, akkor a telepítési folyamat megszakadt és egy hibaüzenet jelenik meg, jelezve, hogy az elérhetetlen Managed Serverek esetleg újra lettek indítva vagy el lettek távolítva a klaszterből, az alkalmazás telepítése előtt.
- Ha egy alkalmazás nem telepíthető klaszterre, akkor csak és kizárólag egyetlen Managed Serverre telepíthető a klaszteren belül (2).

Alapértelmezésben a WebLogic Server támogatja, hogy ne a teljes klaszterre telepítsünk. Ha egy vagy több Managed Server nem elérhető a klaszterből, akkor a következő üzenet jelenik meg: *„Unable to contact "servername". Deployment is deferred until "servername" becomes available."* Amikor a szerver elérhetővé válik, akkor a telepítés azon a szerverpéldányon is elindul. Amíg a telepítési folyamat be nem fejeződik, addig a Managed Server hiányzó vagy érvénytelen osztályok miatt hibákba ütközhet. Ha beállítjuk a ClusterConstraintsEnabled-t, akkor biztosak lehetünk benne, hogy a telepítés csak akkor lesz végrehajtva, ha az összes klaszterbeli Managed Server elérhető. Ha a ClusterConstraintsEnabled értékét igazra állítjuk, akkor a klaszterre való telepítés csak akkor lesz sikeres, ha a klaszter minden tagja elérhető és telepíteni tudja a megadott fájlokat (2).

Sun Java System Web Server 7.0 alkalmazása elosztott rendszerekben

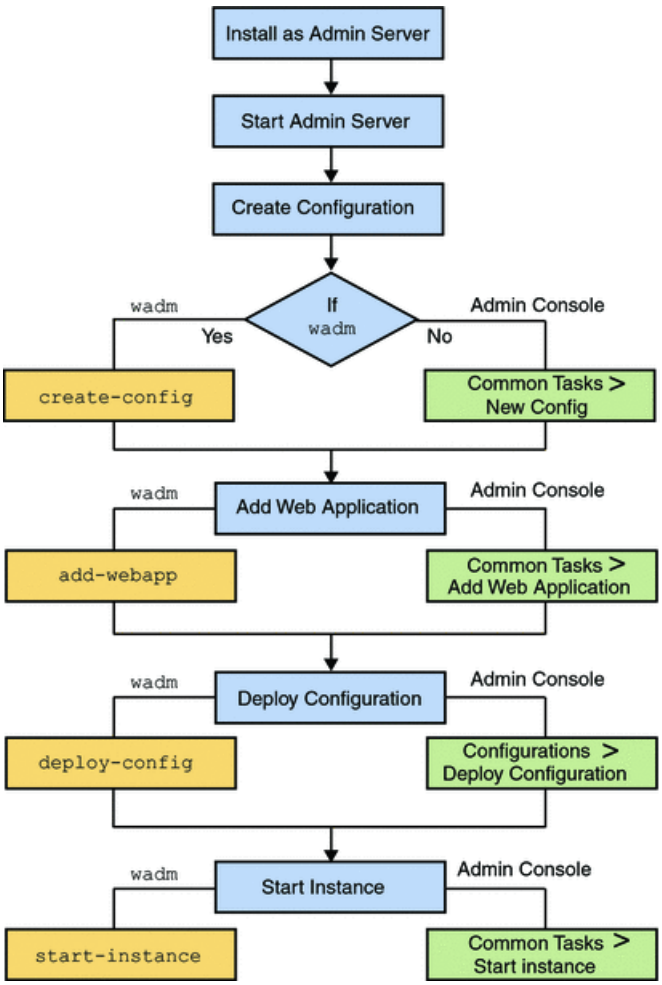
Ha a Web Server 7.0-t egy egyedi csomópontra telepítjük, akkor a Web Server a következő architektúrát fogja követni:



1. **Adminisztrációs szerver:** Ez egy speciálisan konfigurált web szerver példány. Az adminisztrációs szerveren web alkalmazásokat telepíthetünk.
2. **Adminisztrációs csomópont:** Az adminisztrációs csomópont egy szerverfarmon belüli csomóponton vagy szerveren/host-on van telepítve, és képes a távoli adminisztrációs szerverrel kommunikálni. Az adminisztrációs szerveren belüli szerver konfigurációk erre a csomópontra telepíthetők. Minden egyes szerverfarmon belüli adminisztrációs csomópontnak homogén kell lennie. Azaz minden csomópontnak ugyanazt az operációs rendszert kell használnia és ugyanazzal a hardware architektúrával kell rendelkeznie.
3. **Konfiguráció:** Egy konfiguráció egy web szerver példány minden egyes konfigurálható elemére vonatkozik, úgy, mint web alkalmazások, konfigurációs fájlok, keresési indexek. Konfigurációt létre lehet hozni, lehet módosítani és törölni. A web szerver képes az összetett konfigurációk kezelésére. Példányok hozhatók létre egy konfiguráció számára. Egy módosított konfiguráció telepítése frissíti az adott konfiguráció példányát.
4. **Konfigurációs tár:** Ez a fájlrendszer-alapú adattár, ahol minden egyes konfiguráció tárolva van. Fontos, hogy ne szerkesszünk egyetlen fájlt sem a config-store könyvtárban! Az e könyvtáron belüli fájlokat a Sun Java Web Server hozta létre, belső használatra. Ha manuálisan kell szerkesztenünk a konfigurációs fájlt a config-store könyvtárban, akkor telepítsük a konfigurációt a wadm deploy-config paranccsal.
5. **Példány:** Egy példány az adott csomóponton lévő web szerver környezetére vonatkozik, magába foglalva annak konfigurációját, log fájljait és más futás közben

előidézett változásokat, mint pl. zárolt adatbázisok, gyorsítótárak és ideiglenes fájlok. Menedzsment célokra egy példány elindítható, megállítható, újraindítható vagy dinamikusan újrakonfigurálható (3).

A web szerver bevezetése egy csomóponton



A bevezetési folyamat a következő lépésekkel írható le:

- Bevezetés előtti követelmények;
- web szerver telepítése.

Bevezetés előtti követelmények

Hogy bevezethessük a web szervert egy egyszerű csomóponton, a rendszert elő kell készíteni

a következő műveletekkel:

1. installáljuk a web szervert egy csomópontra.

   Ha a gyors telepítési módot választjuk a telepítés során, a következő alapértelmezett egyedek jönnek létre:

   - egy adminisztrációs szerver;
   - egy alapértelmezett konfiguráció egy HTTP listener-rel és egy virtuális szerver; a konfiguráció és a virtuális szerver nevének meg kell egyeznie a host nevével.
   - egy példány az alapértelmezett konfigurációból.

2. Indítsuk el az adminisztrációs szervert (3)!

Az adminisztrációs szerver egy szabványos SSL porton kezd futni (3).

Web szerver bevezetése

Használjuk a következő eljárást a web szerver csomóponton történő bevezetéséhez!

1. Használhatjuk az alapértelmezett konfigurációt vagy létrehozhatunk új konfigurációt. Ha új konfigurációt hozunk létre, akkor a konfigurációnak egyedi nevet kell megadni. Az új konfiguráció egy virtuális szervert és egy alapértelmezett HTTP listener-t hoz létre.
   Ha az adminisztrációs konzolt használjuk egy konfiguráció létrehozására, akkor a varázsló figyelmeztet egy új példány létrehozására. Ha a parancssori felületet használjuk, explicit módon kell létrehoznunk a konfigurációs példányt, a creat-instance parancs használatával.
   Minden konfiguráció a config-store könyvtárban tárolódik a <install_dir>/admin-server/ könyvtár alatt.

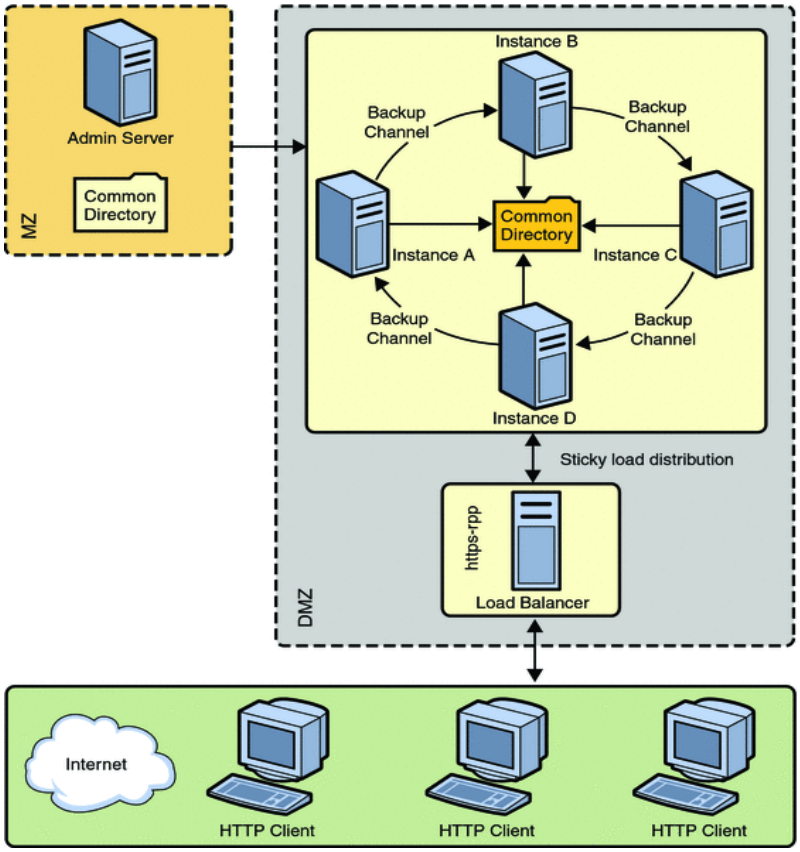
2. Telepítsük a módosított konfigurációt (3).

Klaszter környezet

Tegyük fel, hogy a web server klaszter a következő entitásokat tartalmazza:

1. 4 példány (futás 4 azonos csomóponton)
2. egy adminisztrációs szerver
3. egy inverz proxy a HTTP kérések terheléselosztásához

A köv. ábra ezt a klaszterbe rendezett környezetet szemlélteti.



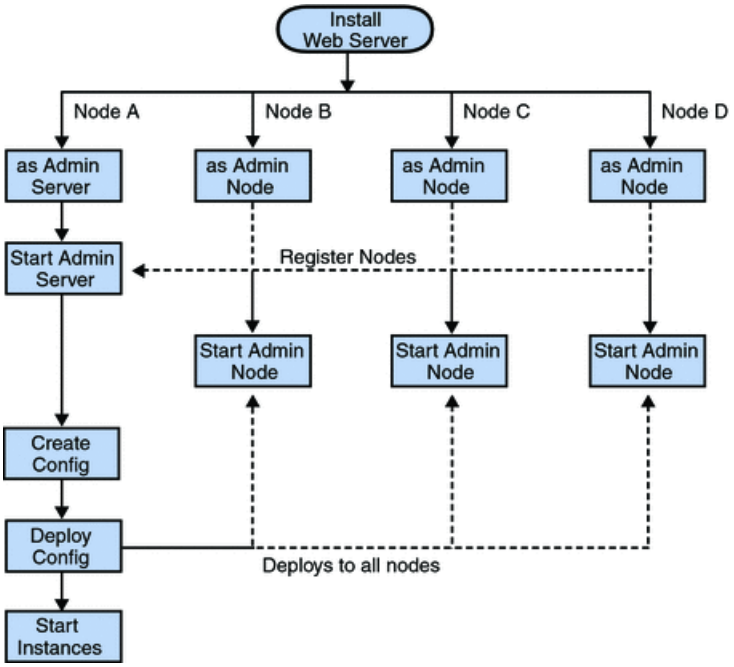
Az ábrán csomópontok vannak konfigurálva a DMZ-ben (demilitarizált zónában). Az adminisztrációs szerver egy tűzfal mögött van konfigurálva (az MZ zónában), hogy korlátozva és védve legyen az adminisztrációs szerver az általános hozzáférésektől. Egy másik csomópont pedig mint inverz proxy-szerver van konfigurálva. Az inverz proxy-szerver a DMZ-n belül tartózkodik, hogy növelje a biztonságot (3).

Hogy beállítsunk egy klasztert 2 vagy több azonos csomópontra van szükségünk ugyanazzal

az operációs rendszer verzióval és ugyanazon telepített patch-ekkel (3).

Klaszter beállítása

A köv. folyamatábra egy klaszter beállítását illusztrálja:



1. az egyik csomóponton telepítsük a web szervert, ami adminisztrációs szerverként funkcionál a klaszteren!
2. a másik 3 csomóponton telepítsük a web szervert! Válasszuk a „web szerver telepítése adminisztrátor csomópontként” opciót! A telepítés során válasszuk a csomópont regisztrálása szerverrel lehetőséget!
3. Bizonyosodjunk meg róla, hogy az adminisztrációs szerver SSL portot használ a kommunikációra, mivel az adminisztrációs csomópontok csak biztonságos módon tudnak regisztrálni a szerveren.
4. Bizonyosodjunk meg róla, hogy a rendszerdátum és idő minden csomóponton, ahol az adminisztrációs szerver és adminisztrációs csomópontok telepítve vannak ugyanaz! A szerver tanúsítványa azon csomópont rendszerdátumán és rendszeridején alapulva kerül létrehozásra, amelyre az adminisztrációs szerver telepítve van. Ha az adminisztrációs csomópont rendszerideje korábbi, mint az adminisztrációs szerveré, a regisztráció hibás lesz, mivel az adminisztrációs szerver tanúsítványa még nem

érvényes.

5. Indítsuk el az adminisztrációs szervert az install_dir/admin-server/bin könyvtárból!

install_dir/admin-server/bin> ./startserv

6. Indítsuk el a wadm parancssor-eszközt az adminisztrációs csomóponton! A wadm parancssor-eszköz az install_dir/bin könyvtáron belül helyezkedik el.

install_dir/bin> ./wadm

7. Regisztráljunk minden adminisztrációs csomópontot az adminisztrációs szerverrel! Használjuk a register-node parancsot, hogy minden csomópontot regisztráljunk a szerverrel!

   pl.:

   ./wadm register-node -user=admin --host=abc.sfbay.sun.com --port=8989,

   ahol abc.sfbay.sun.com az adminisztrációs szerver neve, ahová regisztráljuk a csomópontot; a port az adminisztrációs szerver SSL port száma;

8. Utasítást kapunk, hogy írjuk be a rendszergazda-jelszót. Írjuk be a rendszergazda-jelszót!

   Az adminisztrációs szerverek hitelesítik egymást, az adminisztrációs szerverek megbíznak az adminisztrációs csomópont szerverének hitelesítésében és az adminisztrációs csomópont megbízik a kliens, adminisztrációs szerver által kibocsájtott hitelesítésében.

   Egy adminisztrációs csomópont regisztrációja során, az adminisztrációs szerver generál egy szervertanúsítványt az adminisztrációs csomópontnak, ami aztán letöltődik és telepítésre kerül az adminisztrációs csomóponton.

   A regisztráció csak SSL-en keresztül történhet.

9. Indítsuk el az összes adminisztrációs csomópontot a „startserv” parancs használatával az install_dir/admin-server/bin könyvtárból.
10. Az adminisztrációs konzolt vagy a CLI-t használva hozzunk létre egy új konfigurációt az adminisztrációs szerveren!

    Biztosítsuk a konfigurációs információkat, mint a konfigurációs nevet, HTTP figyelő portot és a szerver nevét az új konfigurációhoz.

11. Hozzuk létre a konfigurációs példányokat minden csomóponton!
12. Indítsuk el a példányokat minden csomóponton (3)!

A terheléselosztást szolgáló inverz proxy konfigurálása

A Web Server 7.0 egy kifinomult beépített terheléselosztót biztosít, az inverz proxy-t. Az inverz proxy egy gateway web szerverek számára a szerverfarmban. Konfigurálva az inverz proxy-t a kérelmek több hasonlóképp konfigurált web szerverhez továbbítódnak.

A Web Server 7.0-n az inverz proxy-t a következőképpen lehet engedélyezni:

1. Telepítsük a web szervert arra a csomópontra, amelyiket az inverz proxy konfigurálására akarunk használni!
2. Hozzunk létre egy konfigurációt! pl.: rp.
3. Használjuk az adminisztrációs konzolt, és válasszuk a Configurations>Virtual Servers>Content Handling>Reverse Proxy fület! Kattintsunk a New gombra!
4. Írjuk be az inverz proxy URI-ját és minden, a klaszterben vesszővel határolt szerver URL-jét! A szerver URL beírásának formátuma: host név:portszám
5. Mentsük el a változásokat!
6. Telepítsük a változtatott konfigurációt, hogy érvényesítsük a változásokat a konfigurációban!
7. Indítsuk el ennek a módosított konfigurációnak minden példányát!

Ezzel kész a HTTP kérések terheléselosztást szolgáló inverz proxy konfigurálása.

AZ ALKALMAZÁS

Amint azt az Oracle WebLogic fejezetben bemutattam az elosztott rendszereknél, amikor elindul egy alkalmazás telepítési folyamata, első lépésben meghatározásra kerülnek a telepíteni kívánt komponensek, valamint a telepítés helye. A programhoz telepíteni kívánt komponensek meghatározásában segíthet ez az alkalmazás, hiszen ha fel szeretnénk telepíteni egy programot, akkor nem kell az időt azzal tölteni, hogy kiderítsük, milyen termékeket kell még beszerezni ahhoz, hogy a program megfelelően működjön. Elég csak elindítani a ProDep-t, kiválasztani a telepítendő terméket és máris láthatjuk a működéséhez szükséges programok listáját.

Felhasználói dokumentáció

A ProDep általános ismertetése

Az alkalmazás segítségével egyszerűen lehet a termékek közötti közvetlen függőségeket nyilvántartani, kezelni és menedzselni. Egy termék előfüggése egy másik terméknek, ha fel kell telepíteni ahhoz, hogy a második termék megfelelően működjön (pl. a .Net Framework 3.5 és a GraphViz is előfeltétele a ProDep-nek).
Egy terméknévhez egy vagy több verzió is tartozhat. Például: .Net Framework 2.0, 3.0, 3.5, 3.5 SP1
Minden verzió a következő adatokkal rendelkezik:

- verziószám,
- befejezési dátum,
- állapot („élő” / „halott”),
- a termékhez írt megjegyzés,
- elő – illetve utófüggő termékek – az is előfordulhat, hogy egy terméknek nincs semmilyen elő – vagy utófüggése.

Az élő vagy halott állapot azt jelenti, hogy a termék adott verziója megvásárolható-e vagy sem. Természetesen a különböző állapotú verziók különbözőképpen jelennek meg az alkalmazásban.
A program műveletei elérhetőek felhasználói menüből vagy a billentyűzetről.
Az alkalmazás 2 szerepkört különböztessen meg: User és Admin, és ezeken belül további

jogosultságok vannak: insert, delete, modify. Ezek a jogosultságok a termékekre vonatkoznak és rendre a következő funkciókat takarják: új termék felvétele, meglévő termék törlése, meglévő termék adatainak módosítása

A ProDep telepítése / eltávolítása

A programot a Setup.msi fájl segítségével lehet telepíteni és eltávolítani.

A program telepítése esetén meg kell adni azt a könyvtárat, ahova a programot telepíteni szeretnénk, majd miután a program működéséhez szükséges fájlok elhelyezésre kerültek, elkezdődik a gráf megrajzolásához szükséges GraphViz program telepítése, ahol szintén meg kell adni a GraphViz célkönyvtárát. Végül meg kell adni annak az adatbázis szervernek a nevét, amelyen el kívánjuk helyezni az adatbázist. Ezután létrejönnek az adatbázistáblák és a tárolt eljárások, ha azok eddig még nem voltak létrehozva. A telepítő automatikusan létrehoz egy adminisztrátort is, melynek felhasználói neve: admin, jelszava: admin111.

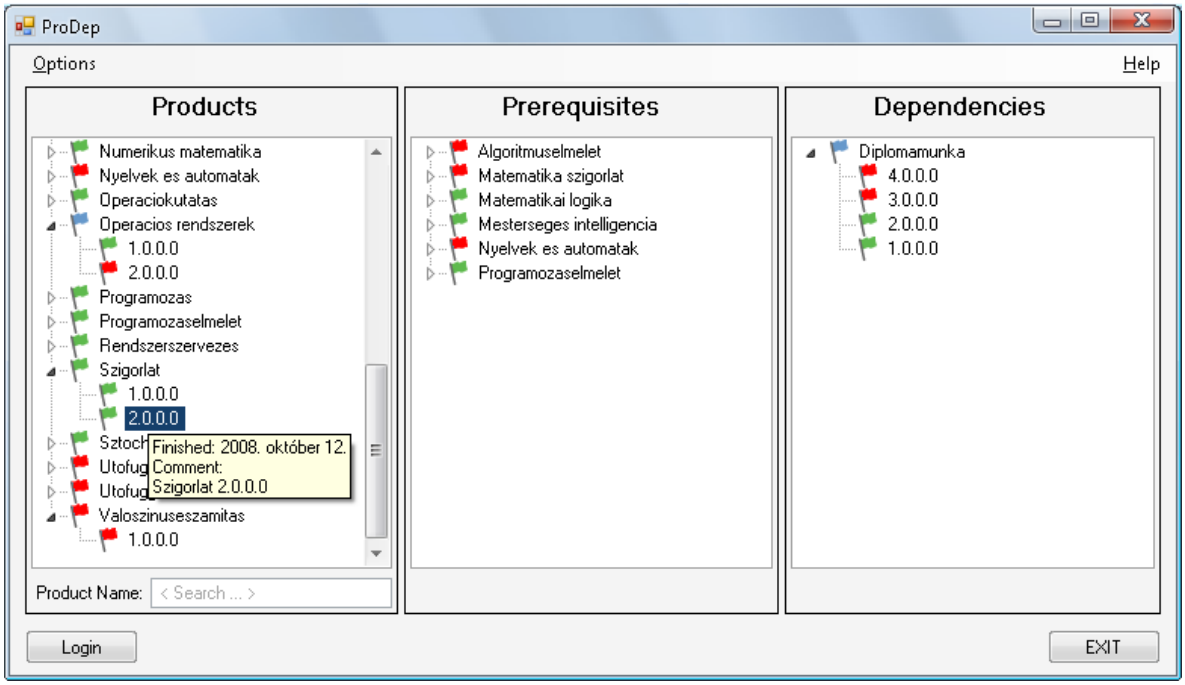
A program eltávolítása esetén 2 lehetőség közül választhatunk. Az egyik lehetőség az esetleges hibák javítása (Repair ProDep), ekkor a program újratelepítése történik meg, míg a másik lehetőség a program eltávolítása (Remove ProDep), ahol a program tényleges eltávolítása történik. A program eltávolításával nem törlődik az adatbázis.

A ProDep felhasználói felülete

A könnyebb érthetőség és átláthatóság érdekében a programban a Debreceni Egyetem Informatikai Karán 2004-ben indult programtervező matematikus szak mintatantervében szereplő tantárgyak szerepelnek.

A „fő” ablak bemutatása

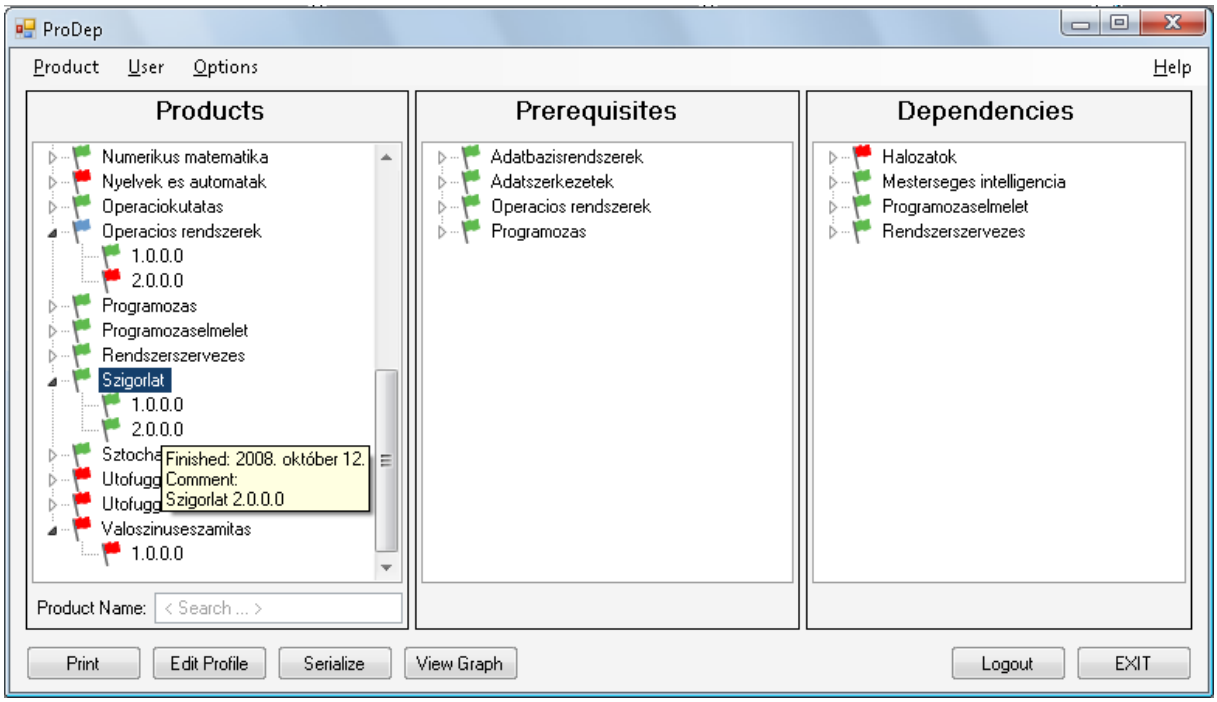
A program indulásakor a következő képernyő fogad minket:



A program elindulásakor csak a termékek közötti közvetlen függőségek megjelenítésére van lehetőség, illetve az Options menü Settings almenüjében be lehet állítani az adatokat tároló adatbázis eléréséhez az adatbázisszerver nevét. A felhasználónak be kell jelentkeznie ahhoz, hogy a termékeket és a felhasználókat menedzselhesse, a termék tetszőleges mélységű függőségeit megjeleníthesse gráf segítségével, a termékeket szerializálhassa, a termék adatait és a közvetlen függőségeit kinyomtathassa. A képen is látható, hogy a termék neve mellett kis zászlók találhatóak. Ezek hivatottak jelezni, hogy egy terméknek milyen verziói vannak (zöld zászló jelzi, hogy a terméknek csak élő verziói vannak, piros zászló jelzi, hogy a terméknek csak halott verziói vannak, a kék zászló pedig azt jelzi, hogy a terméknek van élő és halott verziója is). A Products panelen lévő faszerkezetben található a program által kezelt összes termék neve és a hozzájuk tartozó verziószámok. A Prerequisites és a Dependencies paneleken pedig a Products faszerkezetben kiválasztott termék elő – illetve utófüggései szerepelnek, azaz azok a termékek, amiktől a kiválasztott termék közvetlenül függ, és azok a termékek, amik közvetlenül függnek attól. A Products faszerkezet alatt található szövegdoboz feladata a termékek szűrése. Ha ide begépelünk egy karaktersorozatot, akkor a faszerkezetben

csak azok a termékek fognak megjelenni, amelyek neve ezzel a karaktersorozattal kezdődik. Ha nincs olyan termék, amely a szövegdobozban megadott karaktersorozattal kezdődik, akkor a szövegdobozban megváltozik a szöveg háttere és színe, mint ahogy az alábbi képen is látszik: 

A belépés után a következőképpen módosul az ablak tartalma:



Mint látható megjelent a Product és User menüpont, valamint a Print, Edit Profile, Serialize, View Graph és Logout gombok. A User menüpont és a Serialize gomb csak Admin szerepkörben érhetőek el. A Product és a User menüpont alatt található a New, Edit, Delete almenük. A Product menüpont almenüi attól függően jelennek meg, hogy a felhasználó milyen jogokkal (insert, modify, delete) rendelkezik. Ha a felhasználó nem rendelkezik egyik joggal sem, akkor természetesen a Product menüpont sem jelenik meg.

A **Print** gomb szolgál a kiválasztott termék adatainak (terméknév, verziószám, elő- illetve utófüggések, a befejezés dátuma, a termék élő vagy halott termék-e, a termékhez írt komment) a kinyomtatására.

Az **Edit Profile** gomb segítségével lehetősége van a bejelentkezett felhasználónak, hogy módosítsa a felhasználói nevét, jelszavát és teljes nevét.

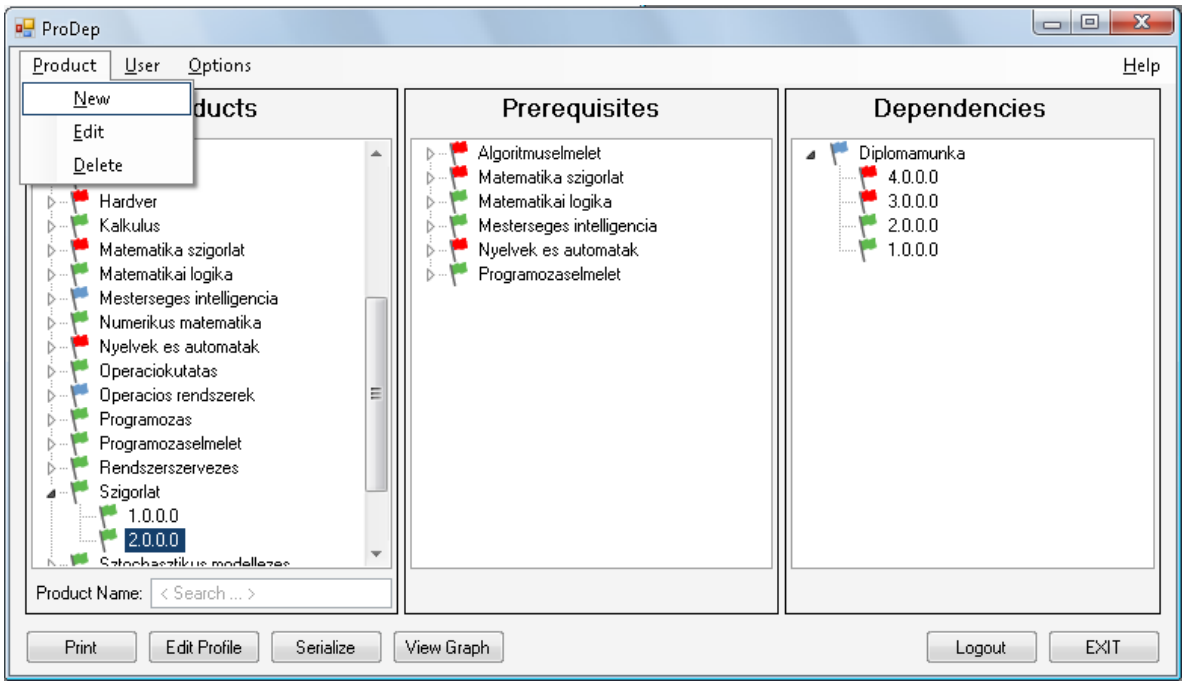
A **Serialize** gomb szolgál a kiválasztott termék adatainak XML formátumban való szerializálására.

A **View Graph** gomb szolgál arra, hogy egy termék függőségeit, tetszőleges mélységig, megjelenítsük egy gráfban. Lehetőség van arra is, hogy az összes termék minden függőségét megjelenítsük. Néhány gráf megtalálható az 1. számú mellékletben.

A **Logout** gomb segítségével tud a bejelentkezett felhasználó kijelentkezni.

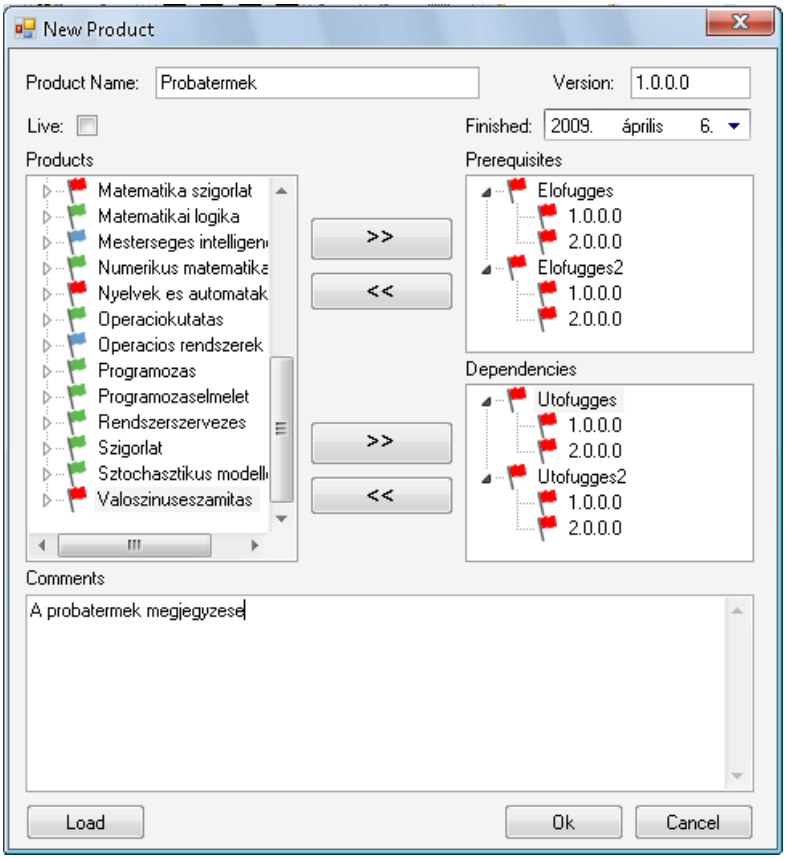
A Product menü bemutatása

A termékek menedzselésére szolgáló menüpont.



A **New** menüpont

Új terméket lehet létrehozni, illetve egy szerializált terméket lehet felvenni a már meglévő termékek mellé. Ez a menüpont aktiválható a Ctrl+N billentyűkombináció segítségével is. Ha ez a menüpont aktiválódik, akkor az alábbi ablak jelenik meg:



Új termék felvételekor meg kell adni:

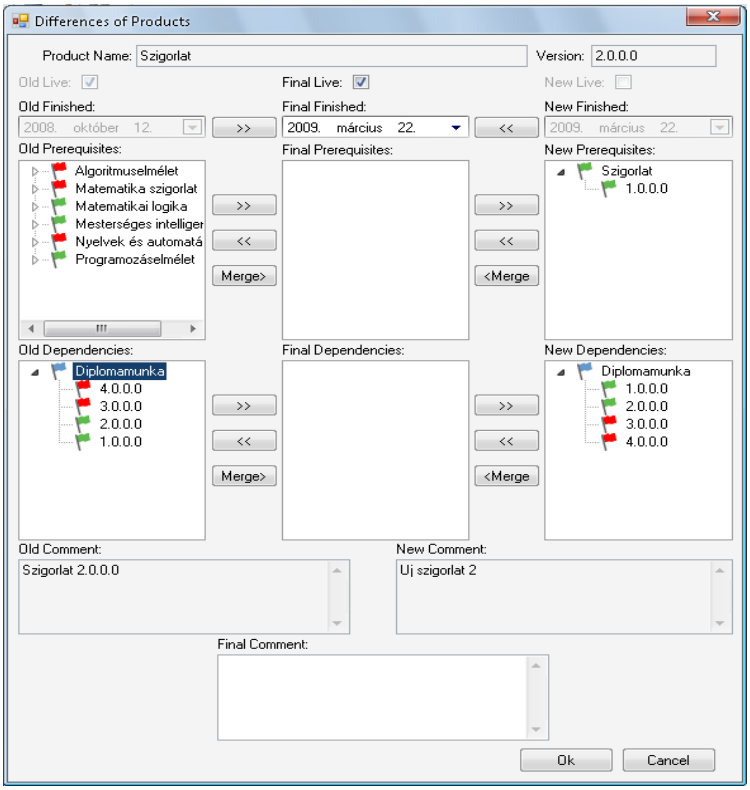
- A termék nevét (Probatermek),
- Verziószámát (1.0.0.0),
- A fejlesztés befejezésének napját (2009. április 6.),
- A termék e verziója élő vagy halott verzió-e (A Live jelölőnégyzet nincs bepipálva ezért a termék halott állapotban lesz létrehozva),
- A termékhez tartozó előfüggéseket (Elofugges 1.0.0.0, Elofugges 2.0.0.0, Elofugges2 1.0.0.0, Elofugges2 2.0.0.0),
- A termékhez tartozó utófüggéseket (Utofugges 1.0.0.0, Utofugges 2.0.0.0, Utofugges2 1.0.0.0, Utofugges2 2.0.0.0) és
- egy opcionális megjegyzést (A probatermek megjegyzese).

A termék neve csak az angol abc betűit és számjegyeket tartalmazhat. A verziószám 4 részből áll, amiket '.'-tal választunk el. Az első 3 rész egy pozitív egész számból áll, míg a 4. rész az egy számból és egy opcionális szövegrészből áll, annak jelölésére, hogy ez a verzió alfa vagy béta állapotban van. A Finished mező értéke alapértelmezés szerint mindig a rendszerdátum. Ha a megjegyzés mező nincs kitöltve, akkor alapértelmezés szerint egy „terméknév verziószám" megjegyzés lesz hozzárendelve.
A **Load** gomb segítségével lehet egy szerializált termék adataival kitölteni a megfelelő mezőket.

Az **Edit** menüpont: A Products faszerkezetben kiválasztott termék adatait lehet módosítani. A menüpont elérhető a Ctrl+E billentyűkombináció segítségével is. Az Edit_Product ablak felépítése annyiban különbözik a New menüpontnál látott ablakétól, hogy itt nincs Load gomb és a mezők természetesen ki vannak töltve a megfelelő adatokkal.

A **Delete** menüpont: A Products faszerkezetben kiválasztott terméket lehet törölni. A menüpont elérhető úgy is, ha a Products faszerkezetben a törölni kívánt termék verziószámán állva megnyomjuk a Delete gombot.

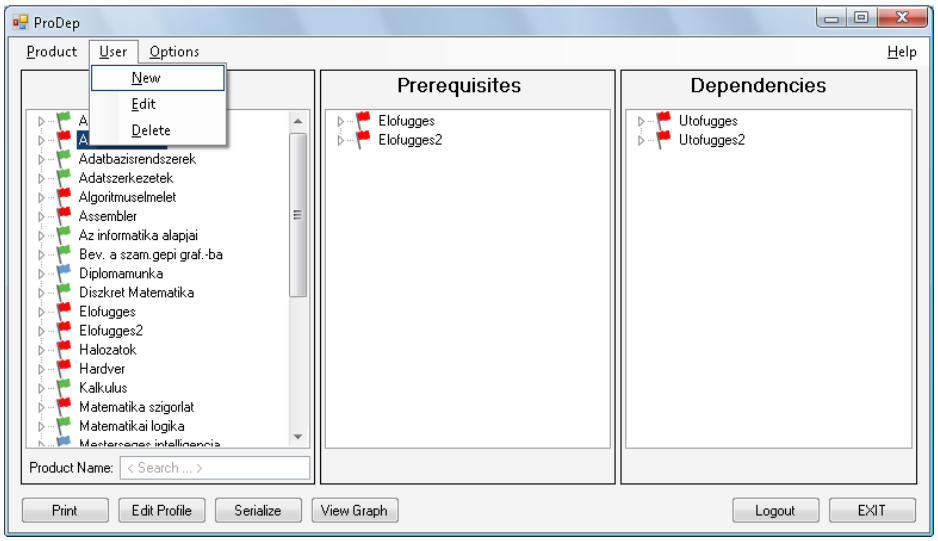
A **termékek közötti különbségek** megjelenítésére akkor kerül sor, ha egy létező termék nevével és verziószámával megegyező névvel és verziószámmal akarunk új terméket létrehozni, vagy egy termék módosításakor a módosítás eredményeként egy ugyanilyen nevű és verziószámú termék áll elő. Ebben az esetben a következő ablak jelenik meg:



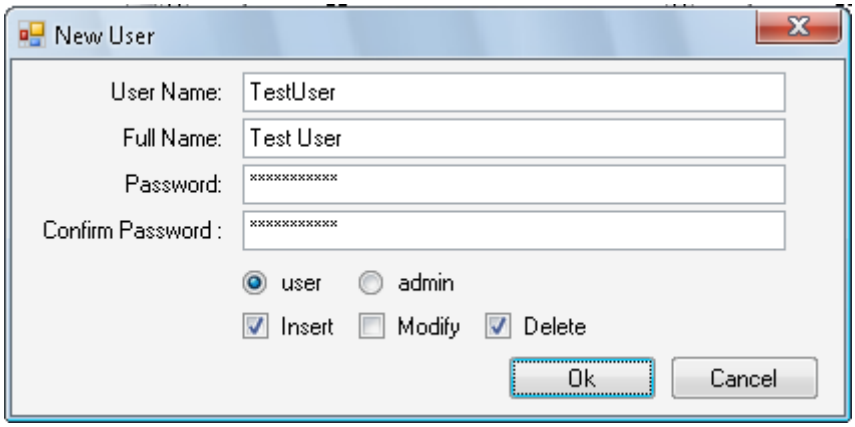
Ezen ablakban bal oldalon az adatbázisban szereplő termék adatai találhatóak, míg jobb oldalon az új termék adatai szerepelnek. Az ablak középső részén találhatóak a végleges termék adatai.

A User menü bemutatása

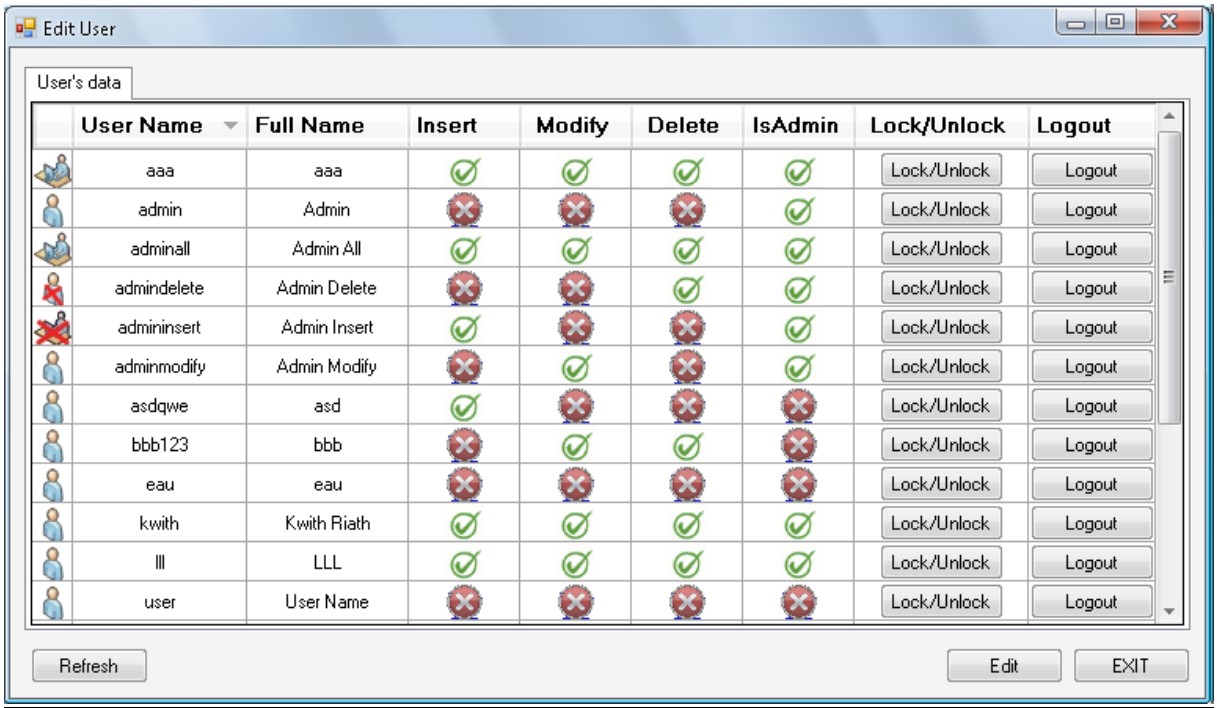
A felhasználók menedzselésére szolgáló menüpont.



A **New** menüpont: Új felhasználó felvételére szolgál. Meg kell adni a felhasználói nevet, egy opcionális teljes nevet, a jelszót, egy megerősítő jelszót, a szerepkört (admin vagy user) és a jogokat (insert, modify, delete).



Az **Edit** menüpont: A felhasználó adatainak módosítására szolgáló menüpont.



Az adminisztrátor ebben az ablakban tudja zárolni a kiválasztott felhasználót. Csak zárolt felhasználót lehet kijelentkeztetni és csak az ő adatai módosíthatóak. Egy adminisztrátor

természetesen csak a saját maga által zárolt felhasználók zárját oldhatja fel és másik adminisztrátor által zárolt felhasználót nem zárolhat, nem jelentkeztethet ki és nem módosíthatja az adatait.

Az ablakban szereplő adatok a Lock/Unlock és Logout oszlopok kivételével bármelyik oszlop szerint rendezhetőek. A Lock/Unlock oszlopban lévő gomb az adott sorban lévő felhasználó zárolására vagy az adminisztrátor által zárolt felhasználó esetén a zár feloldására szolgál. A Logout oszlopban szereplő gomb bejelentkezett és zárolt felhasználók kijelentkeztetésére szolgál.

Az Insert, Modify, Delete oszlopok a jogokat szemléltetik, míg az IsAdmin oszlop szemlélteti, hogy az adott felhasználó user vagy admin szerepkörben van-e. A zöld pipa jelzi az igaz értéket, azaz a joggal való rendelkezést, vagy az admin szerepkört, míg a piros körben található X jelzi, hogy a kiválasztott felhasználó nem rendelkezik az adott joggal vagy user szerepkörrel bír.

Az ablakban 4 fajta ikon jelzi a felhasználók állapotait:

1. Az íróasztal mögött ülő emberalak jelzi, hogy az adott felhasználó be van jelentkezve a programba.
2. Az íróasztal mögött ülő, piros vonalakkal áthúzott emberalak jelzi, hogy az adott felhasználó be van jelentkezve a programba, de éppen zárolva van. Ekkor a felhasználó nem tud semmilyen módosítást végrehajtani.
3. Az emberalak jelzi, hogy a felhasználó nincs bejelentkezve a programba.
4. A piros vonalakkal áthúzott emberalak jelzi, hogy a felhasználó nincs bejelentkezve a programba, de zárolva van. Ebben az esetben a felhasználó nem tud bejelentkezni az alkalmazásba.

A Refresh gomb az adatok manuális frissítésére szolgál. Az ablakban szereplő adatok minden módosítás után frissülnek.

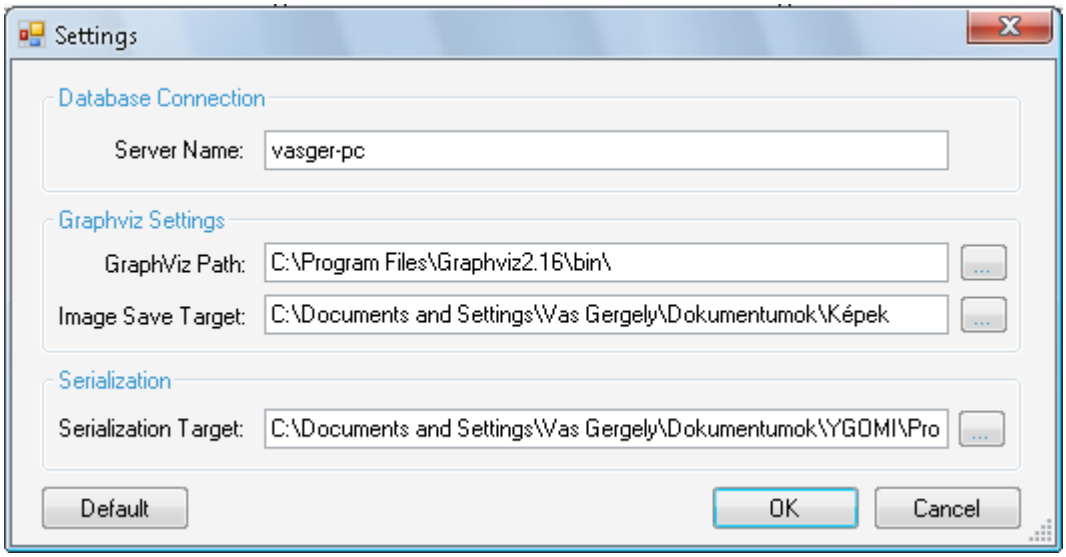
Az Edit gomb segítségével lehet a kiválasztott, zárolt felhasználó adatait módosítani. Az ablak felépítése megegyezik az új felhasználó felvételénél bemutatott ablakkal, kivéve, hogy ha az adminisztrátor a saját adatait akarja módosítani, akkor nem módosíthatja a szerepkörét és a jogait. Ugyanez az ablak jelenik meg, ha az alkalmazásba való bejelentkezés után az Edit Profile gombra kattint a felhasználó.

A **Delete** menüpont: A felhasználók törlésére szolgáló menüpont. Ha ez a menüpont aktiválódik, akkor az Edit menüpontnál ismertetett ablakkal megegyező felépítésű ablak jelenik meg, annyi különbséggel, hogy nem Edit, hanem Delete gomb van az ablakban.

Az Options menüpont

Az Options menüpontban található a fontosabb beállítások elvégzésére alkalmas Settings menüpont.

A Settings menüpont



A Database Connection részben lehet beállítani az adatokat tartalmazó adatbázisszerver nevét. Ez a rész mindig látható.

A GraphViz Settings résznél lehet beállítani a gráf kirajzolásához a GraphViz program bin mappájának az elérési útját valamint a gráfot ábrázoló kép tárolási helyét. A képek jpg formátumban vannak letárolva. Ez a rész csak bejelentkezés után látható.

A Serialization résznél lehet beállítani azt a mappát, ahová a szerializált állományok kerüljenek. Ez a rész csak bejelentkezés után látható az admin szerepkörű felhasználók számára.

A Default gomb segítségével állítható vissza az ablak megnyitásakor életben lévő beállítások.

A ProDep és a naplózás

A programban a bejelentkezést, kijelentkezést, a termékek és a felhasználók menedzselését naplózzuk. A naplózás a felhasználó számítógépén és az adatbázisban is megtörténik.

Fejlesztői dokumentáció

Az osztályok részletes leírása

Product osztály

Adattagok

public string productName: string típusú adattag a termék nevének tárolására.
public ArrayList versionList: ArrayList kollekció a termékhez tartozó verziók tárolására.
public Version[] versions: Version objektumokat tartalmazó tömb.
public ArrayList prerequirements: ArrayList az előfüggő termékek tárolására.
public ArrayList dependencies: ArrayList az utófüggő termékek tárolására.

Metódusok

public Product(): alapértelmezett konstruktor.
public Product(string prodName, string version, string finished, bool live, TreeView preTreeView, TreeView depTreeView, string comment): Konstruktok egy új termék létrehozására a megadott adatok alapján. Ezek az adatok rendre a termék neve, verziószáma, a befejezés dátuma, a verzió állapota, az előfüggőségeket tartalmazó faszerkezet, az utófüggéseket tartalmazó faszerkezet és a verzióhoz írt megjegyzés.
public void AddVersion(Version version): Felvesz egy új verziót a termékhez.
public string getName(): Visszaadja a termék nevét.
public string getVersions(): Visszaadja a termékhez tartozó verziószámokat.
public void Serialize(): Szerializálja a terméket.

public void setProductName(string name): A paraméterben megkapott értékre állítja a termék nevét.
public string compareTo(Product otherProd): Összehasonlít két terméket és visszaadja a két termék közötti különbséget.
public override string ToString(): Visszaadja a termék adatait szövegként.
public override bool Equals(object obj): Megnézi, hogy két termék azonos – e.
public override int GetHashCode(): Visszaadja a termék hash kódját.

Version osztály

Adattagok

public string version: string típusú adattag a verziószám tárolására.
public string finished: string típusú adattag a termék befejezési dátumának tárolására.
public bool live: bool típusú adattag a termék állapotának tárolására.
public string comment: string típusú adattag a termékhez írt megjegyzés tárolására.

Metódusok

public Version(): Alapértelmezett konstruktor.
public Version(string version, string finished, bool live, string comment): Konstruktor egy új verzió létrehozására a paraméterként megkapott verziószámmal, befejezési dátummal, állapottal és megjegyzéssel.
public override string ToString(): A verzió adatait adja vissza szövegként.
public override bool Equals(object obj): Két verzió egyezőségének vizsgálatára szolgáló metódus.
public override int GetHashCode(): A verzió hash kódját adja vissza.

User osztály

Adattagok

private string user_name: string típusú adattag a felhasználói név tárolására
private string full_name: string típusú adattag a felhasználó teljes nevének tárolására
private bool insert: bool típusú adattag az insert jog tárolására
private bool modify: bool típusú adattag a modify jog tárolására
private bool delete: bool típusú adattag a delete jog tárolására.
private bool isadmin: bool típusú adattag, aminek az értéke igaz, ha a felhasználó admin szerepkörben van és hamis ha user szerepkörben van.
private bool loggedIn: bool típusú adattag annak tárolására, hogy a felhasználó be van-e jelentkezve a programba vagy sem.

Metódusok

public User(): Alapértelmezett konstruktor
public User(string user_name, string full_name, bool insert, bool modify, bool delete, bool isadmin, bool loggedIn): Új felhasználót hoz létre a paraméterben megadott adatokból
public string logToString(): A felhasználó adatait adja vissza a naplózáshoz szöveg formában.
public override string ToString(): A felhasználó adatait szövegként visszaadó metódus
public override bool Equals(object obj): Két verzió egyezőségének vizsgálatára szolgáló metódus
public override int GetHashCode(): A verzió objektum hash kódját adja vissza.

Tulajdonságok

public string User_NameProperty: Felhasználói névhez tartozó tulajdonság
public string Full_NameProperty: Teljes nevet reprezentáló tulajdonság
public string InsertProperty: Az insert jogot szemléltető tulajdonság
public string ModifyProperty: A modify joghoz tartozó tulajdonság

public string DeleteProperty: A delete jogot reprezentáló tulajdonság
public string IsAdminProperty: Az admin szerepkör meglétéhez tartozó tulajdonság

Full_Form osztály

Adattagok

Internal láthatóságú adattagok
internal static User user: Az éppen bejelentkezett felhasználót reprezentáló statikus User objektum.
internal Dictionary<string, byte> lockedUser: Az Edit User és Delete User ablakokon a zárolt felhasználók felhasználói nevét és állapotát tartalmazó „szótár”.

Privát láthatóságú adattagok
private int currentPage: A nyomtatásnál az aktuális oldalszámot nyilvántartó int típusú adattag.
private int maxPage: A nyomtatásnál a maximális oldalszámot nyilvántartó int típusú adattag.
private int tmpi,tmpj,tmpk,tmpl,tmpLineNumber: A nyomtatásnál használt ideiglenes adattagok.
private string selectedProductName: A Products faszerkezetben kiválasztott termék nevét tároló string típusú adattag.
private string selectedVersion: A Products faszerkezetben kiválasztott termék verziószámát tároló string típusú adattag.
private int graphDeep: A függőségi gráf mélységét tároló int típusú adattag.
private bool question: A felhasználó saját állapotának ellenőrzésénél használt bool típusú adattag.
private static string logPath: A naplófájl elérési útját tároló statikus string típusú adattag.
private TreeView allProdTV: Az összes terméket tartalmazó fastruktúra.

Tulajdonságok

internal int Deep: A függőségi gráf mélységét reprezentáló csak írható tulajdonság.
internal static bool Refresh: A „fő” ablak frissítésének szükségességét nyilvántartó bool típusú adattag.

Metódusok

public Full_Form(): Alapértelmezett konstruktor, amely ellenőrzi a naplófájl meglétét és ha nincs meg a naplófájl, akkor létrehozza azt.
private void Full_Form_Load(object sender, EventArgs e): A program „fő” ablakának a megjelenítésére szolgáló metódus.
private void Form_Closed(object sender, EventArgs e): Új termék felvétele, meglévő termék törlése vagy egy termék adatainak módosítása esetén frissíteni kell a program által megjelenített adatokat. Ebben játszik szerepet ez a metódus.
internal static string Hash(string input): A paraméterben kapott szöveg MD5 hash kódját visszaadó metódus.
private void ToolStripMenuItem_Products_NewProduct_Click(object sender, EventArgs e): Az új termék felvételét lehetővé tevő ablak megjelenítésére szolgáló metódus.
private void ToolStripMenuItem_Products_DeleteProduct_Click(object sender, EventArgs e): A kiválasztott termék törlésére szolgáló ablak megjelenítésére szolgáló metódus.
private void ToolStripMenuItem_Products_EditProduct_Click(object sender, EventArgs e): A kiválasztott termék adatainak módosítását lehetővé tevő ablak megjelenítésére szolgáló metódus.
private void TreeView_Products_AfterSelect(object sender, TreeViewEventArgs e): Ha kiválasztottunk egy terméket a Products faszerkezetben, akkor annak meg kell jeleníteni az elő – illetve az utófüggéseit. Ez a metódus felelős ezért a funkcióért.
private void ToolStripMenuItem_User_NewUser_Click(object sender,EventArgs e): Az új felhasználó felvételére szolgáló ablak megjelenítésére szolgáló metódus.
private void ToolStripMenuItem_User_EditUser_Click(object sender,EventArgs e): Ez a metódus felelős a felhasználók adatainak módosítását lehetővé tevő ablak megjelenítéséért.

private void EditUser_Closing(object sender, CancelEventArgs e): Mielőtt bezáródik a felhasználói adatok módosításáért illetve a felhasználók törléséért felelős ablak ellenőrizni kell, hogy nincs – e zárolva felhasználó az adminisztrátor által. Ha van, akkor ezt a zárat az ablak bezárásának véglegesítésekor fel kell oldani.

private void ToolStripMenuItem_User_DeleteUser_Click(object sender,EventArgs e): A felhasználók törléséért felelős ablak megjelenítéséért felelős metódus.

private void TextBox_Search_ProductName_MouseClick(object sender, MouseEventArgs e): Amikor belekattintunk a szűrő szövegdobozba, akkor eltűnik a benne található < Search ... > szöveg.

private void TextBox_Search_ProductName_TextChanged(object sender, EventArgs e): A Products faszerkezetben lévő adatok szűréséért felelős metódus.

private void TextBox_Search_ProductName_Leave(object sender, KeyEventArgs e): Ha elhagyjuk a Products faszerkezet tartalmát szűrő szövegdobozt, akkor ez a metódus fut le. Ha a szövegdoboz üres, akkor az visszaáll az eredeti állapotára.

private void Button_Login_Click(object sender, EventArgs e): A bejelentkező képernyő megjelenítéséért felelős metódus.

private void loginForm_Closed(object sender, EventArgs e): A bejelentkező ablak bezárása után azonnal lefutó metódus. Feladata, hogy megjelenítse a bejelentkezett felhasználó szerepkörének megfelelően a gombokat, menüelemeket.

private void Button_Exit_Click(object sender, EventArgs e): Ezzel a metódussal lehet kilépni a programból.

private void Button_Print_Click(object sender, EventArgs e): A nyomtatási kép megjelenítéséért felelős metódus.

private void printDocument_PrintPage(object sender, PrintPageEventArgs e): A kiválasztott termékhez elkészíti a nyomtatónak elküldendő lapot.

private void TreeView:KeyDown(object sender, EventArgs e): Ha valamelyik faszerkezeten vagy gombon állva leütünk valamilyen billentyűt, akkor ez a metódus vizsgálja meg, hogy valamilyen funkcióért felelős billentyűkombináció lett – e leütve vagy sem.

private void Button_EditProfile_Click(object sender, EventArgs e): A felhasználói profil módosításáért felelős ablak megjelenítéséért felelős metódus.

private void Button_Logout_Click(object sender, EventArgs e): A felhasználó kijelentkezik a programból.

internal void Logout_From_Other_Form(object sender, EventArgs e): Ez a metódus akkor fut le,ha a program használata közben a bejelentkezett felhasználót egy adminisztrátor kijelentkezteti.

private void ToolStripMenuItem_Help_About_Click(object sender,EventArgs e): Az AboutBox – t jeleníti meg.

private void Button_Serialize_Click(object sender, EventArgs e): A termék szerializációját indítja el.

private void Button_Graph_Click(object sender, EventArgs e): A függőségi gráf megrajzolását és megjelenítését végző metódus.

private static void Logging(string What, string With, bool fromLoginForm): A naplózást végző metódus.

private void settingsToolStripMenuItem_Click(object sender,EventArgs e): A Settings ablak megjelenítéséért felelős metódus.

internal static int ControllProfile(bool message): A bejelentkezett felhasználó profiljának ellenőrzésére szolgáló metódus, amely azt vizsgálja, hogy zárolták – e a profilt.

internal static void Locking(string What, byte NewState): A zárolást végző osztály szintű metódus.

internal static void MoveTreeNode(TreeView fromTV, TreeView toTV, TreeNode whatTN): Ez a metódus szolgál arra, hogy a fromTV faszerkezet egy whatTN elemét áthelyezze a toTV faszerkezetbe.

New Product osztály

Adattagok

private string differenceDefaultProductAndEditedProduct: Ha a termék módosítása ablakból hívódott meg az új termék példányosítása, akkor ebbe az adattagba kerül be a „fő” ablakban kiválasztott termék és a módosítás eredményeként előálló termék közötti különbség.

private Product primaryProductFromFullForm: A „fő” ablakban kiválasztott terméket reprezentáló Product objektum.

private bool question: Annak eldöntésére szolgáló adattag, hogy az OK gombra kattintáskor kell – e megerősítést kérni vagy sem. Ha a „fő” ablakban az új termék hozzáadása lett

aktiválva, akkor kell megerősítés, míg ha termék módosításánál kell az új terméket felvenni, akkor nem kell megerősítés.

private TreeView defaultProductsTV: Ez a faszerkezet típusú adattag megegyezik a „fő” ablak Products faszerkezetével.

private Full_form fullFrom: A „fő” ablakot reprezentáló Full_form objektum. Feladata, hogy a felhasználói profil vizsgálatakor, ha ki lett jelentkeztetve a felhasználó, akkor meg kell hívni a fullForm Logout_From_Other_Form metódusát.

Metódusok

public New_Product(): Alapértelmezett konstruktor.

public New_Product(Product productFromFullForm, Product editedProduct, Full_form ff): A termék módosítása ablakból ezzel a konstruktorral lehet példányosítani az osztályt. Az ff lesz a fullForm adattag értéke, az editedProduct adataival kitöltjük az ablakot, és a productFromFullForm és az editedProduct termékek közötti különbség által visszaadott string lesz a differenceDefaultProductAndEditedProduct adattag értéke.

public New_Product(TreeView products, Full_form ff): A „fő” ablakból ezzel a konstruktorral lehet példányosítani az osztályt. Ez a konstruktor fut le akkor, ha a „fő” ablakon a New Product menüpontot választotta, vagy Ctrl+N-t billentyűkombinációt ütött a felhasználó.

public void Button_Ok_Click_From_Edit_Product(object sender, EventArgs e): Ha a termék módosítása ablakból példányosítottuk az osztályt, akkor valahogy el kell érni, hogy aktivizálódjon az új termék ablak OK gombjára való kattintás, anélkül, hogy a felhasználó látná az ablakot. Ezt teszi lehetővé ez a metódus.

private void Button_Cancel_Click(object sender, EventArgs e): A Cancel gombra kattintáskor megerősítést kér a program a felhasználótól. Ha a felhasználó megerősítette a bezárási szándékát, akkor bezáródik az ablak és nem történik meg az új termék felvétele.

private void Differences_of_Products_Form_Closed(object sender, EventArgs e): Ez a metódus szolgál arra, hogy ha a két termék közötti különbözőséget megjelenítő ablak bezáródásával az új termék ablak is bezáródjon.

private Button_AddPrerequirements_Click(object sender, EventArgs e): Ez a metódus fut le akkor, ha egy új előfüggést adunk meg az új terméknek.

private Button_RemovePrerequirements_Click(object sender, EventArgs e): Ez a metódus fut le, ha az új termék egy előfüggést eltávolítja a felhasználó.
private Button_Add_Dependencies_Click(object sender, EventArgs e): Ez a metódus fut le, ha a felhasználó egy új utófeltételt ad az új termék eddigi utófeltételeihez.
private Button_Remove_Dependencies_Click(object sender, EventArgs e): Ez a metódus fut le akkor, ha a felhasználó egy utófüggést eltávolít az eddigiek közül.
private void TextBox_ProductName_KeyDown(object sender, EventArgs e): Ha valamelyik kontrolon állva az Esc billentyűt leütjük, akkor meg kell hívni a Cancel gombra kattintást kezelő metódust, hogy elkezdődjön az ablak bezárása.
private void Button_OK_Click(object sender, EventArgs e): Az új termék felvételét kezelő metódus. Ha az új termék neve és verziószáma már szerepel az adatbázisban, akkor megjelenik a két termék közötti különbözőséget kezelő ablak.
private void Load_Click(object sender, EventArgs e): Egy szerializált termék adatainak betöltéséért felelős ez a metódus.

Edit_Product osztály

Adattagok

private TreeView prodTV: Az összes terméket tartalmazó faszerkezet.
private bool question: Annak a nyilvántartására szolgál, hogy kell – e megerősítést kérni a program használójától.
private Product productFromFullForm: A „fő” ablakban a Products faszerkezet kiválasztott elemét reprezentáló Product objektum. Azaz azt a terméket jelenti, amelyiknek módosítani akarjuk az adatait.
private Full_form fullForm: A „fő” ablakot reprezentáló Full_form objektum. Feladata, hogy a felhasználói profil vizsgálatakor, ha ki lett jelentkeztetve a felhasználó, akkor meg kell hívni a fullForm Logout_From_Other_Form metódusát.

Metódusok

public Edit_Product(Product productFromFullForm, TreeView fullFormProdTV,

Full_form ff): Ezzel a konstruktorral lehet példányosítani az osztályt a „fő” ablakból. A paraméterben megkapott termék a „fő” ablakban a Products faszerkezetben kiválasztott terméket reprezentálja. A fullFormProdTV az összes terméket tartalmazó faszerkezet, míg az ff az maga a „fő” ablak.

private void Edit_Product_Load(object sender, EventArgs e): Az ablak megjelenítéséért felelős metódus.

private void Button_AddPrerequirementes_Click(object sender, EventArgs e): Az összes nem függő terméket tartalmazó faszerkezetből az aktuálisan kiválasztott terméket áthelyezi a módosítás eredményeként előálló termék előfeltételei közé.

private void Button_RemovePrerequirementes_Click(object sender, EventArgs e): A módosítás eredményeként előálló termék előfeltételei közül egy terméket áthelyez az összes nem függő terméket tartalmazó faszerkezetbe.

private void Button_AddDependencies_Click(object sender, EventArgs e): Az összes nem függő terméket tartalmazó faszerkezetből az aktuálisan kiválasztott terméket áthelyezi a módosítás eredményeként előálló termék utófeltételei közé.

private void Button_RemoveDependencies_Click(object sender, EventArgs e): A módosítás eredményeként előálló termék utófeltételei közül egy terméket áthelyez az összes nem függő terméket tartalmazó faszerkezetbe.

private void Edit_Product_KeyDown(object sender, KeyEventArgs e): A billentyűleütéseket figyelő metódus.

private void Button_Default_Click(object sender, EventArgs e): Az alapállapotba hozásért felelős metódus.

private void Button_Cancel_Click(object sender, EventArgs e): A Cancel gombra kattintásért felelős metódus.

private void Button_OK_Click(object sender, EventArgs e): Az OK gombra kattintást és az új termék felvételét kezelő metódus.

private void New_Product_Form_Closed(object sender, EventArgs e): Ha felvettük a módosítás eredményeként előálló terméket az adatbázisba, akkor ezt az ablakot is bezárhatja a program.

Delete_Product osztály

Adattagok

private Full_form fullForm: A „fő” ablakot reprezentáló Full_form objektum. Feladata, hogy a felhasználói profil vizsgálatakor, ha ki lett jelentkeztetve a felhasználó, akkor meg kell hívni a fullForm Logout_From_Other_Form metódusát.

Metódusok

public Delete_User(): Az alapértelmezett konstruktor.
public Delete_User(string Product_Name, string Version, string fullcomment, bool live, string finished, TreeView TreeView_PreRequisites, TreeView TreeView_Dep, Full_form ff): A „fő” ablakból ezzel a konstruktorral lehet példányosítani az osztályt. A paraméterként kapott adatokkal tölti ki az ablakot.
private void Button_OK_Click(object sender, EventArgs e): A „fő” ablakban kiválasztott termék törlésének elvégzése.
private void Button_Cancel_Click(object sender, EventArgs e): Ha az ablakon a Cancel gombra kattintunk, akkor megerősítést kér a program a felhasználótól. Ha a felhasználó megerősítette a bezárási szándékát, akkor az ablak bezáródik és a termék törlése nem történik meg.
private void Delete_Product_KeyDown(object sender, KeyEventArgs e): Ha valamelyik kontrolon állva az Esc billentyűt leütjük, akkor meg kell hívni a Cancel gombra kattintást kezelő metódust, hogy elkezdődjön az ablak bezárása.

Differences_of_Products osztály

Adattagok

private Product oldProductInDB: Az adatbázisban szereplő termék
private Product primaryProductFromFullForm: Új termék felvétele esetén az értéke null, míg termék adatainak módosításánál a „fő” ablakban kiválasztott terméket reprezentálja.

private Full_form fullForm: A „fő” ablakot reprezentáló Full_form objektum. Feladata, hogy a felhasználói profil vizsgálatakor, ha ki lett jelentkeztetve a felhasználó, akkor meg kell hívni a fullForm Logout_From_Other_Form metódusát.

Metódusok

public Differences_of_Products(): Az alapértelmezett konstruktor.

public Differences_of_Products(Product newProduct, Product dbProduct, Product primaryProductFromFullForm, Full_form ff): Az új termék és a termék módosítása ablakból ezzel a konstruktorral példányosítható az osztály. A newProduct objektum az új vagy a módosítás eredményeként előálló termék, a dbProduct az adatbázisban lévő ugyanolyan nevű és verziószámú termék, a primaryProductFromFullForm új termék felvétele esetén null, míg termék módosítása esetén a „fő” ablakban kiválasztott termék, az ff pedig maga a „fő” ablak.

private void Button_Cancel_Click(object sender, EventArgs e): Ha az ablakon a Cancel gombra kattintunk, akkor megerősítést kér a program a felhasználótól. Ha a felhasználó megerősítette a bezárási szándékát, akkor az ablak bezáródik az adatbázisban a régi termék marad benne, míg az új termék elvész.

private void Button_OldPre_to_FinalPre_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha az adatbázisban lévő termék előfeltételei közül egyet áthelyezünk a végleges termék előfeltételei közé.

private void Button_FinalPre_to_OldPre_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha a végleges termék előfeltételei közül egyet áthelyezünk az adatbázisban lévő termék előfeltételei közé.

private void Button_Merge_OldPre_FinalPre_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha az adatbázisban lévő termék összes előfeltételét áthelyezzük a végleges termék előfeltételei közé.

private void Button_OldDep_to_FinalDep_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha az adatbázisban lévő termék utófeltételei közül egyet áthelyezünk a végleges termék utófeltételei közé.

private void Button_FinalDep_to_OldDep_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha a végleges termék utófeltételei közül egyet áthelyezünk az

adatbázisban lévő termék utófüggések közé.

private void Button_Merge_OldDep_to_FinalDep_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha az adatbázisban lévő termék összes utófeltételét áthelyezzük a végleges termék utófeltételei közé.

private void Button_NewPre_to_FinalPre_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha az új termék utófeltételei közül egyet áthelyezünk a végleges termék utófeltételei közé.

private void Button_FinalPre_to_NewPre_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha a végleges termék előfeltételei közül egyet áthelyezünk az új termék előfeltételei közé.

private void Button_Merge_NewPre_FinalPre_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha az új termék összes előfeltételét egyszerre helyezzük át a végleges termék előfeltételei közé.

private void Button_NewDep_to_FinalDep_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha az új termék utófeltételei közül egyet áthelyezünk a végleges termék utófeltételei közé.

private void Button_FinalDep_to_NewDep_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha a végleges termék utófeltételei közül egyet áthelyezünk az új termék utófeltételei közé.

private void Button_Merge_NewDep_to_FinalDep_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha az új termék összes utófeltételét egyszerre helyezzük át a végleges termék utófeltételei közé.

private void Button_NewFinished_to_FinalFinished_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha az új termék befejezési dátumát átmásoljuk a végleges termék befejezési dátumának.

private void Button_OldFinished_to_FinalFinished_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha az adatbázisban lévő termék befejezési dátumát átmásoljuk a végleges termék befejezési dátumának.

private void Button_Ok_Click(object sender, EventArgs e): Ez a metódus akkor fut le ha rákattint a felhasználó az OK gombra. Módosítja az adatbázisban lévő termékhez tartozó adatokat.

New_User osztály

Adattagok

private Full_form fullForm: A „fő” ablakot reprezentáló Full_form objektum. Feladata, hogy a felhasználói profil vizsgálatakor, ha ki lett jelentkeztetve a felhasználó, akkor meg kell hívni a fullForm Logout_From_Other_Form metódusát.

Metódusok

public New_User(): Az alapértelmezett konstruktor.
public New_User(Full_form ff): A „fő” ablakról ezzel a konstruktorral lehet példányosítani az osztályt.
private void Button_OK_Click(object sender, EventArgs e): Az OK gombra kattintást kezelő metódus. Ha az OK gombra kattint a felhasználó, akkor a kitöltött adatok alapján felvesz egy új felhasználót az adatbázisba.
private void Button_Cancel_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha a Cancel gombra kattint a felhasználó. Ekkor megerősítést kér a program és ha a felhasználó megerősíti a bezárási szándékát, akkor az ablak bezáródik és az új felhasználó nem lesz létrehozva.
private void TextBox_UserName_KeyDown(object sender, KeyEventArgs e): Ez a metódus akkor fut le, ha valamelyik vezérlőelemen állva billentyűleütés történik. Ha a leütött billentyű az Esc, akkor meghívódik a Cancel gombra kattintást kezelő metódus.

Edit_User osztály

Adattagok

private static User editedUser: A módosított felhasználót reprezentáló User objektum.
private static User selectedUser: A kiválasztott felhasználót reprezentáló User objektum.
private int columnIndex: Azon oszlop indexe, amely alapján rendezzük az adatokat.
private bool asc: Az oszlop rendezésének az iránya.

private Full_form fullForm: A „fő” ablakot reprezentáló Full_form objektum. Feladata, hogy a felhasználói profil vizsgálatakor, ha ki lett jelentkeztetve a felhasználó, akkor meg kell hívni a fullForm Logout_From_Other_Form metódusát.

Tulajdonság

public static User EditedUser: A módosított felhasználó adatait tartalmazó editedUser objektumot kezelő tulajdonság.

Metódusok

public Edit_User(bool isEditUser, Full_form ff): A „fő” ablakból ezzel a konstruktorral lehet példányosítani az osztályt. A paraméterben szereplő isEditUser értéke igaz, ha a felhasználókat szerkeszteni akarja a felhasználó, azaz az Edit User menüpont aktivizálódott és hamis, ha a felhasználók közül törölni akar a program használója. Az ff pedig maga a „fő” ablak.

private void Edit_User_Load(object sender, EventArgs e): Az ablak megjelenítéséért felelős metódus.

private void Button_Cancel_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha a Cancel gombra kattint a felhasználó. Ekkor megerősítést kér a program, és ha a felhasználó megerősíti a bezárási szándékát, akkor az ablak bezáródik és az új felhasználó nem lesz létrehozva.

private void Button_Edit_Click(object sender, EventArgs e): Ez a metódus akkor fut le, ha felhasználó szerkesztése esetén az Edit gombra, felhasználó törlése esetén a Delete gombra kattintunk.

private void editSelectedUserForm_Closed(object sender, EventArgs e): Ez a metódus akkor fut le, ha befejeztük a felhasználó adatainak a szerkesztését.

private void ListView_UserName_MouseDoubleClick(object sender, MouseEvent-Args e): Ez a metódus, akkor fut le, ha valamelyik lista nézetben a kiválasztott felhasználóra duplán kattint az adminisztrátor. Ekkor is megjelenik a felhasználó adatainak módosítását lehetővé tevő ablak.

private void ListView_UserName_KeyDown(object sender, KeyEventArgs e): A lista

nézetben a leütött billentyűket vizsgáló metódus.

private void dataGridView1_ColumnHeaderMouseClick(object sender, DataGridView-CellMouseEventArgs e): A táblanézet oszlopfejlécére kattintást és ezáltal az adott oszlop alapján történő rendezést kezelő metódus.

private void dataGridView1_CellMouseDoubleClick(object sender, DataGridViewCellMouseEventArgs e): Ha egy sor valamely adatot tartalmazó elemére duplán kattintunk, akkor azt úgy kezeljük, mintha az Edit vagy a Delete gombra kattintott volna a felhasználó.

private void dataGridView1_CellClick(object sender, DataGridViewCellEventArgs e): A táblanézet valamely sorában a Lock / Unlock vagy a Logout gombokra kattintást kezelő metódus.

private void ToolStripMenuItem_Lock_Click(object sender, EventArgs e): A lista nézetben a gyorsmenü Lock / Unlock menüpontjára kattintást kezelő metódus.

private void Button_Lock_Click(object sender, EventArgs e): A Lock / Unlock gombra való kattintást kezelő metódus.

private void ToolStripMenuItem_Logout_Click(object sender, EventArgs e): A lista nézetben a gyorsmenü Logout menüpontjára kattintást kezelő metódus.

private void Button_Logout_Click(object sender, EventArgs e): A Logout gombra való kattintást kezelő metódus.

private void Button_Refresh_Click(object sender, EventArgs e): Az adatok frissítésére szolgáló gombra kattintást kezelő metódus.

RowComparer osztály

Az osztály implementálja az IComparer interfészt.

Adattagok

private static int sortOrderModifier: A rendezés irányát befolyásoló int típusú adattag.

private int coulmnIndex: Azon oszlop indexe, amelyik alapján rendezni kell a táblanézetet.

Metódusok

public RowComparer(SortOrder sortOrder, int columnIndex): Ezzel a konstruktorral lehet példányosítani az Edit_User / Delete_User ablakból ezt az osztályt. Paraméterben megkapja a rendezés irányát és azon oszlop indexét, amely alapján rendezni kell az adatokat.
public int Compare(object x, object y): Az IComparer interfész Compare metódusának implementálása. A metódus két táblanézet sort hasonlít össze a columnIndex adattagban tárolt indexű mező értéke alapján.

Edit_Selected_User osztály

Adattagok

private User defaultUser: Azon felhasználót reprezentáló User objektum, amely adatait módosítani szeretnénk.
private Full_form fullForm: A „fő" ablakot reprezentáló Full_form objektum. Feladata, hogy a felhasználói profil vizsgálatakor, ha ki lett jelentkeztetve a felhasználó, akkor meg kell hívni a fullForm Logout_From_Other_Form metódusát.

Metódusok

public Edit_Selected_User(): Az alapértelmezett konstruktor.
public Edit_Selected_User(User user, bool visible, Full_form ff): Ezzel a konstruktorral lehet példányosítani az osztályt a „fő" és a felhasználókat felsoroló ablakból.
private void Button_Cancel_Click(object sender, EventArgs e): A Cancel gombra kattintás kezeléséért felelős metódus.
private void Button_OK_Click(object sender, EventArgs e): Az OK gombra kattintás kezeléséért, azaz a felhasználó adatainak módosításáért felelős metódus.
private void Edit_Selected_User_KeyDown(object sender, KeyEventArgs e): A billentyűleütések figyeléséért felelős metódus.
private void Edit_Selected_User_SizeChanged(object sender, EventArgs e): Az ablak méretének változását kezelő metódus.

Login_Form

Adattagok

private bool login: Annak a reprezentálására szolgáló bool típusú adattag, hogy az adott user nevű és jelszavú felhasználó be van-e jelentkezve a programba vagy sem.

Metódusok

public Login_Form(): Alapértelmezett konstruktor.
private void Button_OK_Click(object sender, EventArgs e): Az OK gombra kattintást kezelő és a bejelentkeztetés végző metódus.
private void Button_Cancel_Click(object sender, EventArgs e): A Cancel gombra kattintást kezelő metódus.
private void Login_Form_KeyDown(object sender, KeyEventArgs e): A billentyűleütéseket figyelő metódus. Ha az Esc billentyűt ütötték le, akkor kilépünk az ablakból, ha az Entert, akkor pedig megpróbál belépni a program ha a user név és a jelszó is meg van adva.

SettingsForm

Metódusok

public SettingsForm(): Alapértelmezett konstruktor. Ekkor minden adat megjelenik az ablakban.
public SettingsForm(int visible): Konstruktor, amely a paraméterben megkapott értékek alapján engedélyezi az ablakban lévő adatok megjelenítését. Ha a paraméterben kapott érték 0 akkor csak az adatbázis szerver nevét lehet módosítani, 1 esetén megjelenik a szerializációval kapcsolatos rész is.
private void SettingsForm_Load(object sender, EventArgs e): Az ablak megjelenítéséért felelős metódus.
private void Button_Target_Click(object sender, EventArgs e): A gráfot tartalmazó képfájl

és a szerializált állományok tárolási helyének a beállítására szolgáló metódus.

private void Button_OK_Click(object sender, EventArgs e): Az OK gombra kattintást kezelő és a beállítások véglegesítését elvégző metódus.

private void Button_Cancel_Click(object sender, EventArgs e): A Cancel gombra kattintást és az ablak bezárását kezelő metódus.

private void Button_Default_Click(object sender, EventArgs e): A Default gombra kattintást és ezáltal az ablak alapállapotba történő visszaállítását végző metódus.

private void SettingsForm_SizeChanged(object sender, EventArgs e): Az ablak átméretezését kezelő metódus.

WhatDeepIsTheGraph

Adattagok

private Full_form fullForm: A „fő” ablakot reprezentáló Full_form objektum. Feladata, hogy a felhasználói profil vizsgálatakor, ha ki lett jelentkeztetve a felhasználó, akkor meg kell hívni a fullForm Logout_From_Other_Form metódusát.

Metódusok

public WhatDeepIsTheGraph(): Alapértemezett konstruktor.

public WhatDeepIsTheGraph(Full_form full_Form): A „fő” ablakról ezzel a konstruktorral lehet példányosítani az osztályt.

private void Button_Cancel_Click(object sender, EventArgs e): A Cancel gombra kattintást és ezáltal az ablak bezárását kezelő metódus.

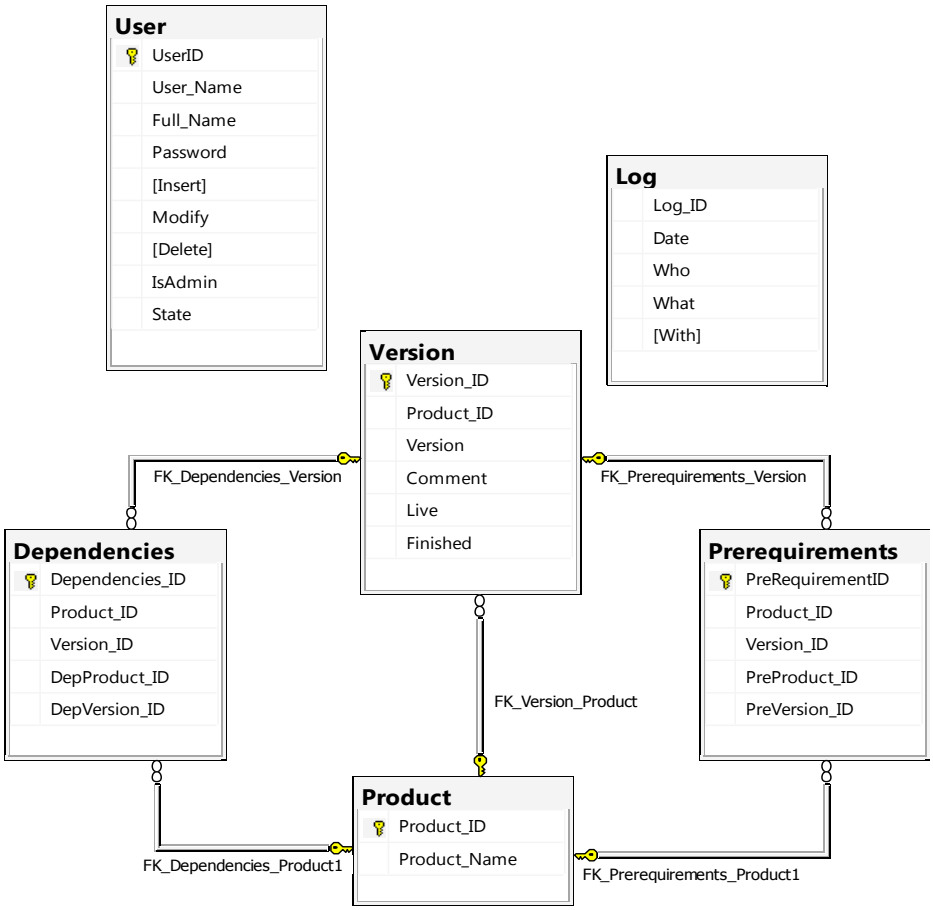
private void Button_Ok_Click(object sender, EventArgs e): Az OK gombra kattintást kezelő és a kirajzolandó gráf mélységének a beállítását elvégző metódus.

private void NumericUpDown_Deep_ValueChanged(object sender, EventArgs e): Ha változtatjuk a mélységet, akkor a rádiógomb aktívvá válik.

private void GraphKeyDown(object sender, KeyEventArgs e): A billentyűleütést vizsgáló metódus. Ha az Esc billentyűt leütjük, akkor bezáródik az ablak és nem lesz kirajzolva a gráf.

Adatbázis oldal

Adatbázisterv



A tárolt eljárások:

DeleteAllDependency: Törli egy termék összes elő – és utófüggését.

Paraméterei: @ProdName, @Version – azon termék neve és verziószáma, amely függőségeit el kell távolítani.

DeleteProduct: Egy termék vagy termékcsalád törlése az adatbázisból

Paraméterei: @ProdName, @Version – azon termék neve és verziószáma, amely terméket töröljük. Ha a @Version értéke *, akkor az adott terméknévhez tartozó összes verzió törlődik az adatbázisból.

DeleteUser: Adott felhasználói névvel rendelkező felhasználó törlése.

Paramétere: @UserName – azon felhasználói név, amelyhez tartozó felhasználót töröljük.

InsertPrerequirement: Adott termékhez lehet beszúrni egy új előfeltételt.

Paraméterei: @ProdName, @Version, @PreProdName, @PreVersion. A @ProdName és a @Version azt a terméket határozzák meg, amihez be kell szúrni az új előfeltételt, míg a @PreProdName és @PreVersion paraméterek a beszúrandó előfeltételt határozzák meg.

InsertProductAndVersion: Egy új terméket szór be az adatbázisba.

Paraméterei: @ProdName, @Version, @Comment, @Live, @Finished. A paraméterek a következőket jelentik: terméknév, verziószám, a termékhez írt megjegyzés, a termék állapota és a befejezés dátuma.

InsertUserProc: Egy új felhasználót szúr be az adatbázisba.

Paraméterei: @UserName, @FullName, @Password, @Insert, @Modify, @Delete, @IsAdmin. A paraméterek a következőket jelentik: felhasználói név, teljes név, jelszó, új termék létrehozásának a joga megvan-e, megvan-e a termék adatainak módosításához a jog, megvan-e a jog a termék törléséhez, admin szerepkörű-e az új felhasználó.

IsProductVersionInDb: Annak eldöntése szolgáló tárolt eljárás, hogy egy termék szerepel-e már az adatbázisban.

Paraméterei: @ProdName, @Version. A két paraméter a kérdéses termék nevét és verziószámát tartalmazza.

Locking: Adott felhasználó névvel rendelkező felhasználó zárolását és kijelentkeztetését végző tárolt eljárás.

Paraméterei: @Who, @UserName, @NewState.

A paraméterek értékei: Ki hajtotta végre a zárolást vagy kijelentkeztetést, milyen felhasználói névvel rendelkező felhasználón, mi az új állapot.

Logging: A naplózást megvalósító tárolt eljárás.

Paraméterei: @LogUserName,@What,@With. A paraméterek értékei: Milyen felhasználói nevű felhasználó, milyen műveletet hajtott végre, mi lett a művelet eredménye.

LoginProc: A paraméterben megkapott felhasználói nevű és jelszavú (MD5-tel titkosítva) felhasználó bejelentkeztetése.

Paraméterei: @UserName, @Password

LogoutProc: A paraméterben megkapott felhasználói nevű felhasználó kijelentkeztetése.

Paramétere: @UserName

ShowDependencies: A paraméterben megkapott termék utófüggéseinek megjelenítésére szolgáló tárolt eljárás.

Paraméterei: @ProdName, @Version

ShowPrerequirements: A paraméterben megkapott termék előfüggéseinek megjelenítésére szolgáló tárolt eljárás.

Paraméterei: @ProdName, @Version

ShowProductAndVersion: A paraméterben megkapott nevű és verziószámú termék adatainak a megjelenítésére szolgáló tártolt eljárás.

Paraméterei: @ProdName, @Version

ShowUserData: A paraméterben megkapott felhasználói névvel rendelkező felhasználó adatainak a megjelenítéséért felelős tárolt eljárás. Ha a paraméter értéke *, akkor az összes felhasználó adatát jeleníti meg.

Paramétere: @UserName

UpdateProduct: A paraméterben megkapott nevű és verziószámú termék állapotát, befejezésének dátumát, megjegyzését módosító tárolt eljárás.

Paraméterei: @ProdName, @Version, @Live, @Finished, @Comment

UpdateUser: A paraméterben megadott felhasználói nevű felhasználó adatait módosító tárolt eljárás.

Paraméterei: @OldUserName, @NewName, @NewFullName, @Password, @NewInsert, @NewModify, @NewDelete, @NewIsAdmin

UserIsInDb: Annak eldöntésére szolgáló tárolt eljárás, hogy van-e az adatbázisban felhasználó, melynek felhasználói neve megegyezik a paraméterben megkapott értékkel.

Paramétere: @UserName

A program esetleges továbbfejlesztési lehetőségei

A programban a mostaninál jobban ki lehetne használni a .Net Framework 3.0-val és az azutáni verziókkal érkező újdonságokat.

Az adatbázisban el lehetne tárolni a programok telepítéséhez szükséges fájlokat, így a programból közvetlenül tudnánk telepíteni a kiválasztott terméket az ő összes előfüggésével együtt.

Ha készítenénk a programnak egy megfelelő honlapot, akkor nem kellene minden kliens gépére külön feltelepíteni a ProDep-et és a GraphViz-t, hanem elég lenne csak a szerverre és így az alkalmazást használó cég többi telephelyéről is, amik akár nagyon távol is elhelyezkedhetnek egymástól, könnyen és gyorsan elérhetővé válnának az adatok.

A felhasználói élményen sokat lehetne javítani, ha a programot átültetnénk Windows Presentation Foundation alapokra, a honlapos megvalósítás esetén Silverlight alapokra.

A honlapos megvalósításnál az Entity Framework tökéletesen alkalmazható lehetne.

KÖSZÖNETNYÍLVÁNÍTÁS

Ez úton szeretnék köszönetet mondani Hajdu Tamásnak és dr. Végh Jánosnak, hogy elvállalták a témavezetésemet, Kern Péternek a program fejlesztése során felmerülő problémák megoldásában nyújtott segítségéért, ötleteiért és Balla Anettnek, Fülöp Istvánnak, Jakab Tamásnak segítségükért.
Szeretnék továbbá köszönetet mondani páromnak, barátaimnak és családomnak, akik az egyetemen töltött éveim alatt végig támogattak, bíztattak és a nehéz időkben is mellettem álltak.

IRODALOMJEGYZÉK

A diplomamunka megírásához használt irodalom

(1) Andrew S. Tannenbaum – Maarten van Steen: Elosztott rendszerek – Alapelvek és paradigmák, Panem kiadó, 2004

(2) http://e-docs.bea.com/wls/docs100/cluster/config.html#wp1043974

(3) http://docs.sun.com/app/docs/doc/819-2629/gehhu?a=view

http://technet.microsoft.com/en-us/library/bb891786.aspx

A program megírásához használt irodalom

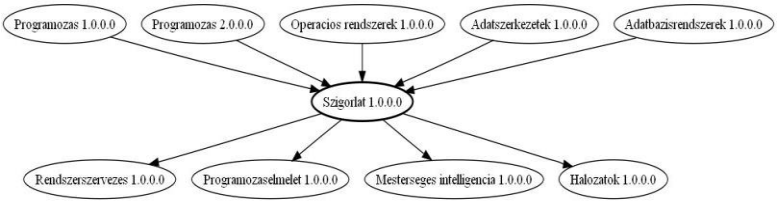
John Sharp: Microsoft® Visual C#® 2005 lépésről lépésre, Szak kiadó, 2005

Kotsis – Légrádi – Nagy – Szénási: Többnyelvű programozástechnika, Panem kiadó, 2007

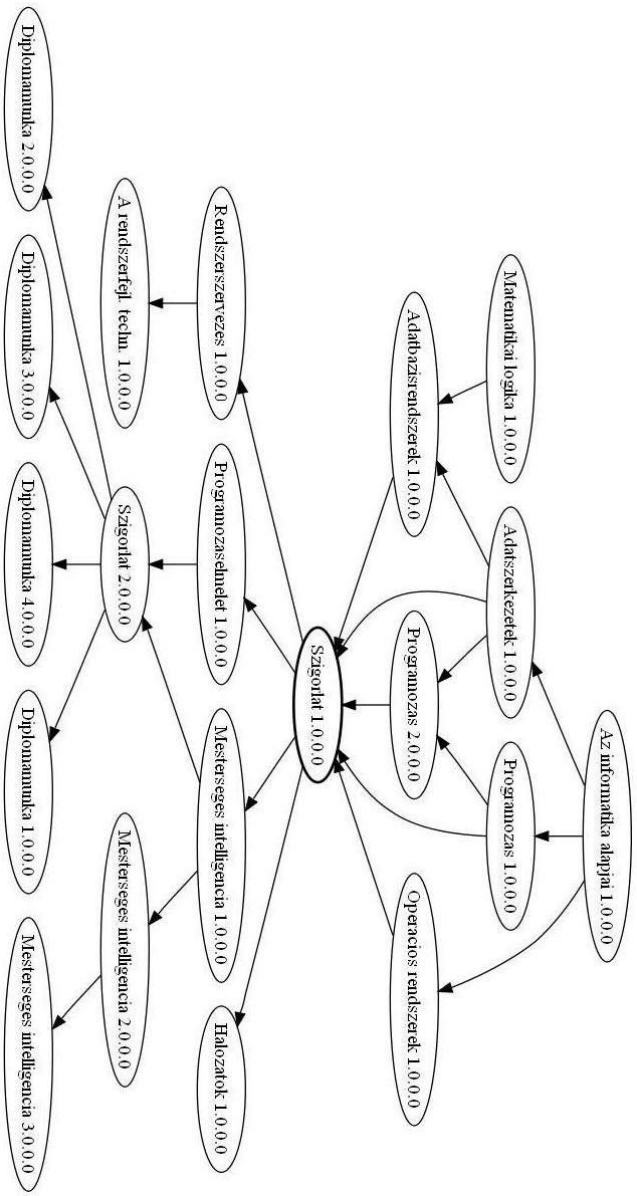
http://msdn.microsoft.com/hu-hu/library/default(en-us).aspx

http://www.koders.com/

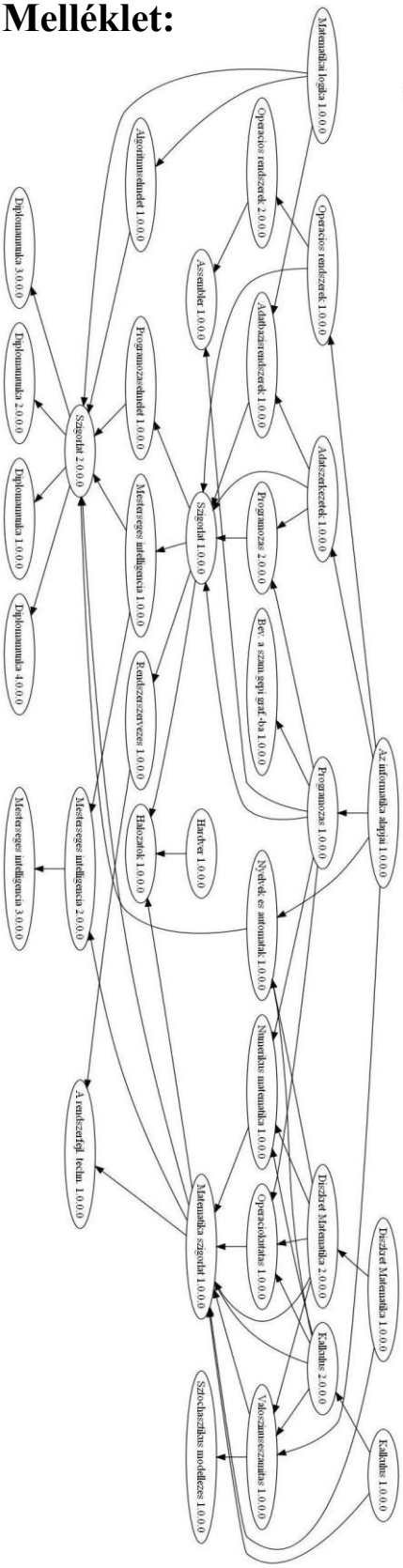
Melléklet:



A Szigorlat 1 közvetlen függőségei



A Szigorlat 1 függőségei 3 szint mélységig



A teljes tanulmányi gráf

1.számú melléklet